A DISCRETE UTTERANCE VOICE RECOGNITION SYSTEM

BY

JAMES F. KEPLER

B.S., University of Illinois, 1984

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1985

Urbana, Illinois

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

THE GRADUATE COLLEGE

JUNE 1985

WE HEREBY RECOMMEND THAT THE THESIS BY

JAMES F. KEPLER

ENTITLED A DISCRETE UTTERANCE VOICE RECOGNITION SYSTEM

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR

THE DEGREE OF MASTER OF SCIENCE

Director of Thesis Research

Head of Department

Committee on Final Examination†

Chairperson

† Required for doctor's degree but not for master's.

O-517

DEDICATION

In memory of my Jennifer, who will always be an inspiration to me.

ACKNOWLEDGMENT

Many people assisted me with this thesis. Without their help and encouragement, it is doubtful that this voice recognition system would have been completed. Thanks are extended to two students, Trang Nguyen and Bill Wigger, for their contributions to the software. I would also like to thank my thesis advisor, Professor Ricardo Uribe, for his advice and encouragement, and for the use of his laboratory, the Advanced Digital Systems Laboratory. Lastly, I would like to thank Tom Liu for introducing me to the topic of voice recognition.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Voice recognition has been a topic of active research for the past 20 years. The applications for a machine that can reliably understand spoken words are numerous. For example, Texas Instruments is working on a voice recognition system to be used in air force planes. This system will enable the pilot to give certain commands to the plane without having to touch any controls. Other companies have made specialized recognition systems for quality control purposes. Here, an inspector on the assembly line simply states his observations as he makes them. The system would record these observations and even take appropriate action, such as removing a defective unit from the line.

In general, there are two types of voice recognition systems: the continuous speech recognition system and the discrete utterance recognition system. The continuous speech

recognizer allows the system to understand entire sentences, spoken in a normal manner (i.e., with no abnormal pauses between words). One of the many problems with this type of system is the determination of the beginning and end of a word within the sentence. However, once some of the key words are determined, context and English grammar rules can be used to help determine other words in the sentence. The discrete utterance recognizer, on the other hand, can not use grammar rules to help in recognizing words, since this type of system looks at each word separately. The system presented in this thesis is the Advanced Digital Systems Laboratory's (ADSL) discrete utterance voice recognition system.

ADSL's research on voice recognition was initiated in 1983 by Thomas Liu, for his Master's of Science degree [1]. The author of the present thesis had the good fortune of working with Mr. Liu on ADSL's first recognition system. After the completion of Mr. Liu's degree, the author of the present thesis continued to work with voice recognition, developing ADSL's second recognition system.

In the present work, four simplifying constraints are made in order to set a realistic goal. As mentioned before, only one word at a time will be recognized. Secondly, the system will not be designed as a real time system, which means that the user will have to wait while the machine recognizes the spoken word. Consequently, no data compression algorithms are used, and the system architecture is not designed for maximum speed. The third simplifying constraint is that the problem of background noise is not treated here. All testing and evaluation of the system are

done in quiet surroundings. Lastly, the system is intended to be a speaker dependent system. Thus, each person who uses the system must train it to his/her particular voice.

Keeping these simplifying constraints in mind, the goal of the thesis can now be stated. Using inexpensive, off-the-shelf hardware, it is desired to build a system that will improve the first ADSL system through hardware developments that will allow for more sophisticated software experiments.

1.1 Background on speech production

Humans can create speech sounds by exciting the vocal tract with some sort of disturbance (e.g., vibration of the vocal cords). This causes the acoustic cavity (vocal tract with or without the nasal cavity) to resonate at certain frequencies, called formant frequencies. These frequencies are determined by the size and shape of the acoustic cavity. It is difficult to classify speech sounds according to the size and shape of the acoustic cavity; however, we can classify speech sounds according to the type of disturbance that caused them. Following this train of thought, there are three general ways in which humans can create speech sounds [2].

1) The vocal tract is excited with a short duration periodic pulse train (i.e., quasi-periodic pulses). These pulses are created by forcing air through constricted vocal cords, thus causing them to vibrate with an amplitude that changes according to a periodic buildup and release of pressure. An example would be

the "oo" sound in the word "good." Generally, any sound that comes from deep within the vocal tract is of this category.

2) The acoustic cavity is excited by a turbulent flow of air created at some point of constriction in the vocal tract. This generates an incoherent excitation for the vocal tract. Examples from this category are the "s" sound and the "sh" sound.

3) The acoustic cavity is excited by the abrupt release of a pressure buildup at some point of closure in the vocal tract. The excitation can include or not include vocal cord vibrations. For example, the "t" in "toe" does not include vocal cord vibrations, while the "b" in "boat" does. This type of excitation can be considered as a step function input to the vocal tract, and is thus a wide band signal.

Another way to categorize speech sounds is whether they are voiced or unvoiced. The sounds that are produced by vocal cord vibrations are referred to as voiced speech sounds. Thus, the first and sometimes the third way of producing speech sounds, as mentioned above, refers to voiced speech. Sounds that are not produced by vocal cord vibrations are termed unvoiced speech sounds. The second and sometimes the third way of producing speech sounds, as mentioned above, fits into this category.

A third way to classify speech sounds is to identify phonemes. A phoneme is the basic linguistic unit which has the property that if one replaces another in an utterance, the meaning of the utterance is changed. The English language can be

broken down into 42 different phonemes.

1.2 An approach to voice recognition

At first glance, it might seem that a discrete utterance recognizer would not be hard to build. For example, a system could be built that identifies which phonemes are in the utterance. After finding out which phonemes are in the utterance, it is a simple matter to determine the spoken word. However, the identification of phonemes is not a simple matter. When a word is uttered, there are no gaps or pause intervals to separate phonemes. The speech sounds are coupled together by a nearly continuous motion of the vocal tract. Therefore, determining the beginning and end of each phoneme is one problem. Also, the English language allows for a certain amount of freedom in specifying a particular phoneme. This freedom can cause problems when trying to identify phonemes within an utterance. For these and other reasons (e.g., the number of comparisons or calculations needed for this approach), the system described in this thesis does not try to identify phonemes.

Another way to approach the recognition problem is to try to determine which type of disturbance is exciting the vocal tract at every instant during the utterance. Since there are only three general types of excitation, this approach needs fewer calculations. Moreover, since the three types of excitation are distinct, there is no problem telling the difference between them. Also, there is no easy way to look at just the excitation of the vocal tract. What can be examined is the output of the

acoustic system due to an excitation. At this point, it is appropriate to introduce a simple model of the vocal system.

Speech can be modeled as the response of a linear time-varying system, the vocal tract, to the appropriate input [3]. If the shape of the vocal tract did not change with time, the output would simply be the convolution of the input with the system response. However, this is not the case, because in forming different sounds, the shape of the vocal tract is changed thereby changing the system impulse response. The result is a system transfer function that changes with time. In figure 1, the output of a stationary vocal tract (i.e., a constant transfer function) is shown when the input is a pulse train.

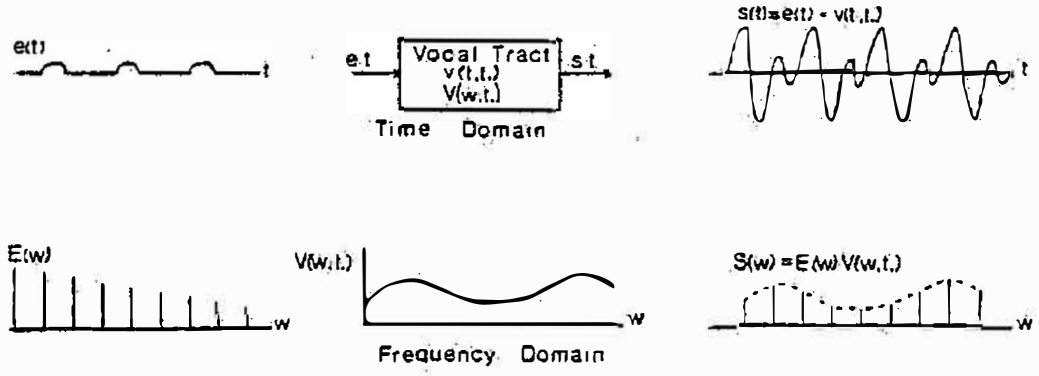


Figure 1 - Stationary vocal tract.

Looking at the system in the frequency domain, as the vocal tract changes, so does the system response, V(w,t). It is clear, then, that the output would be greatly affected by variations in the vocal tract. However, the vocal tract changes slowly with time [4]. Therefore the output, on a short time basis, is the convolution of the input and a constant system response. The

result is that a periodic input will result in a periodic output, thereby allowing the determination of such an input by looking at the output. Also, if the input to the system is incoherent and has most of its energy at the higher frequencies, then the output will generally be incoherent with most of its energy at the higher frequencies. This incoherent input corresponds to the second general way that humans can create speech sounds. Consequently, it is possible to determine this type of input by examining the output of the vocal tract. Finally, if the input is an abrupt wide band signal, the output can be expected to be an abrupt wide band signal. This type of input corresponds to the third way that humans can create speech, and as stated, it is possible to determine this type of input by looking at the output.

Although it is clear that the type of input can be determined, this by itself will not be enough information to recognize words. For example, if each word is characterized by the types of inputs, both of the words "see" and "she" would be characterized as 2,1. Here the 2 means that the first part of the word was uttered using the second way that humans can create speech sounds (an incoherent excitation of the vocal tract). The 1 means that the second part of the word was uttered using the first way to create speech sounds (a quasi-periodic excitation). Clearly, this characterization produces the same code for many different words and thus will not work by itself. However, if information about the input frequency and the formant frequencies is also included, then unique characterizations are possible for many different words. Fortunately, it is possible to get a good

idea of the input and formant frequencies by looking at the output of the vocal tract. The author concludes that the knowledge of both, the type of vocal tract excitation and the frequency content at the output of the vocal tract, is enough information with which to recognize words. This is the information the recognition system looks for.

CHAPTER 2

SYSTEM OVERVIEW

Figure 2 shows the general configuration of the recognition

Figure 2 - General system configuration.

system described in this work. The feature extractor is a microprocessor controlled hardware circuit that is responsible for gathering specific information from the microphone signal. The information is passed to the host computer via a single 9600 baud, full duplex, serial channel. The host processes the information to determine what word was uttered.

The choice of a serial link over a parallel link is made because the 8751 has an on board serial port (more on this in section 3.3). The split architecture, that is, the division of the system into a feature extractor and a host computer, was

chosen for two reasons. First, it keeps the gathering of information and the processing of information clearly separate. This means that both parts of the system can be debugged and developed separately. Secondly, this type of architecture has many advantages for experimentation. Generally, it is desirable to program everything in high level languages so that experimental changes can be made quickly. However, for gathering the information, the system must have fast reactions, necessitating the use of machine level programming. Therefore, the data gathering is programmed in machine language and is part of the feature extractor. The data processing, which is more likely to be experimented with, is programmed in a high level language (turbo pascal) for ease of experimentation.

2.1 Feature extractor

The function of the feature extractor is to capture the essential information from the voice signal. As mentioned previously, the information sought is the type of vocal tract excitation and the frequency content at the output of the vocal tract. To extract this information, the approach used is to split up the voice signal using bandpass filters (BPFs). It is known that the frequency content of speech is from 100 Hz to 3000 Hz, and thus the filters must cover at least this range. Following the direction of past research done in ADSL, it was decided to use seven bandpass filters, each filter an octave apart, covering the range 62.5 Hz to 4000 Hz [1]. The filters are designed with center frequencies: 62.5 Hz, 125 Hz, 250 Hz,

500 Hz, 1000 Hz, 2000 Hz, 4000 Hz.

Since the speech signal is broken into different frequency bands, little information is lost by connecting each filter output to a peak detector (PD). A peak detected signal need not be sampled as often as the original signal. This allows a single A/D converter (with a multiplexer on its input) to handle all seven bands, thereby reducing the hardware complexity. Also, past research indicates that zero crossings can provide useful information [5]. Therefore, the output of each filter, as well as the microphone signal, is connected to a zero crossing (ZC) detection circuit. Figure 3 shows a block diagram of the feature extractor. In summary, the information available from the feature extractor is the waveform envelopes in the seven bands, the zero crossings in each band, and the zero crossings for the microphone signal.

The operation of the feature extractor is as follows: The host computer sends a command to the 8751 microcontroller to start looking for a word. The 8751 starts sampling the zero crossing counters and the peak detectors at regular intervals. When the microcontroller determines that the beginning of a word has been found, it saves the data it takes in and sends it serially to the host computer. Since the serial channel is not fast enough to keep up with the rate of data input, there is a data buffer memory included in the feature extractor. The 8751 also determines when the end of a word has been reached. As mentioned before, the microcontroller determines the system sample rate, i.e., the "regular interval," which is currently set at 10 ms. The determination of the system sample rate will be

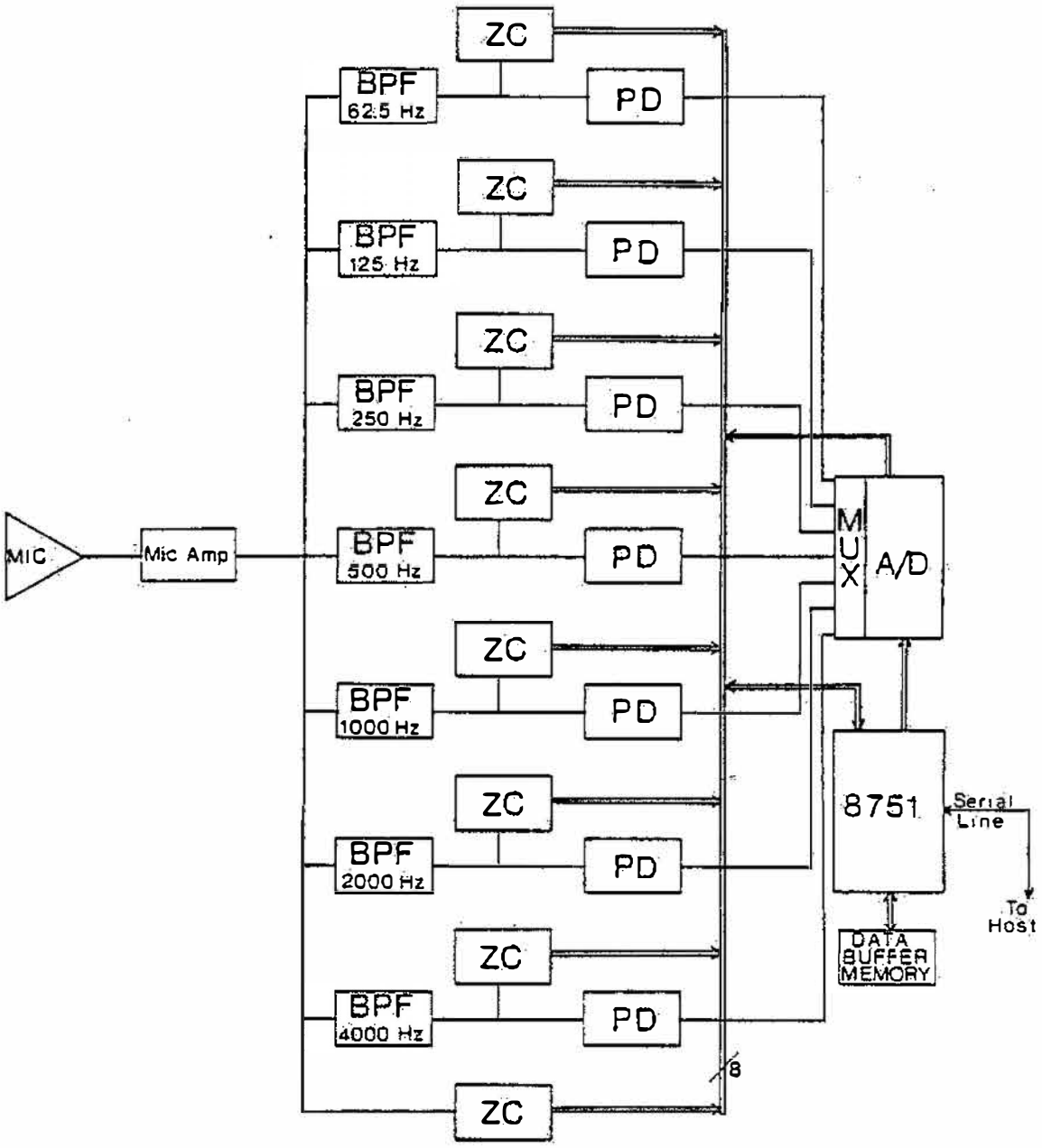


Figure 3 - Feature extractor.

discussed in section 4.1 of this thesis. During the sampling process, data are collected from the peak detectors and then from the zero crossing counters. This group of data is called a frame. A frame can be thought of as a "snapshot" of the feature extraction at a particular time. Since the A/D and the zero crossing counters all have 8 bits of resolution, each frame consists of 15 bytes (7 PD + 8 ZC, see figure 3). However, the 8751 immediately encodes each byte into two ascii hexadecimal bytes, causing the frame length to be 30 bytes instead of 15 bytes. The conversion will be discussed in Chapter 4.

At first glance, it might seem that the information given by the feature extractor is not consistent with the desired information, but this is not the case. For example, consider the output of the vocal tract when it is excited by quasi-periodic pulses. If the voice signal is passed through a microphone, the signal can be plotted on a voltage vs time graph, as shown in figure 4 [6]. Figure 5 shows this same signal after

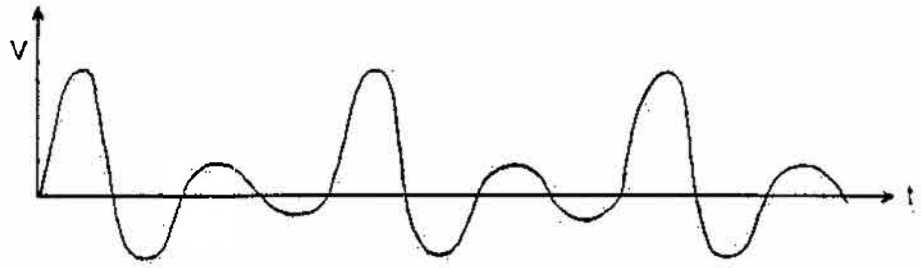


Figure 4 - Vocal tract output.

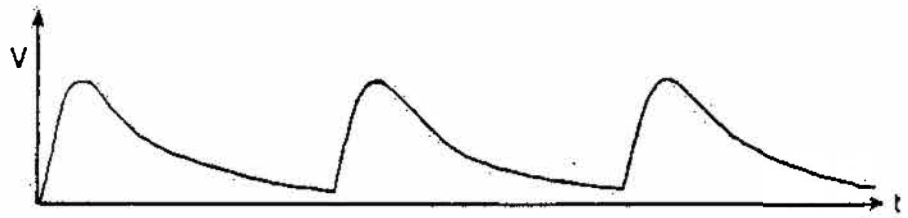


Figure 5 - Peak detected vocal tract output.

it has been peak detected. During the course of the current work, it was confirmed that this type of pattern is distinguishable from other patterns. Thus, by looking at the patterns, it can be determined if quasi-periodic pulses excited the vocal tract. However, this is only part of the desired information. The band in which this pattern shows up will indicate the general frequency content of the waveform. Also, the zero crossings will indicate which part of the band the signal occupies. Thus, the identification of the band in which the pattern was found, and the zero crossings in that band, indicate the frequency content at the output of the vocal tract, which is the other part of the desired information.

2.2 Host Computer

Since the feature extracting hardware is designed with a serial interface, almost any computer can be a candidate for the host computer. The host chosen for this thesis is the Zenith Z-151 desk-top computer. The Z-151 is an IBM PC compatible

computer, having 320K bytes of main memory, a 10 megabyte hard disk, and a 16 bit CPU [7]. As mentioned previously, the software for the host computer is written in turbo pascal. This computer language is chosen because it compiles quickly and because the first recognition system's software was written in pascal. Thus, the differences between the recognition software of the two systems can be easily seen by inspecting the programs.

The host computer has two major functions. First, during the training phase, the Z-151 stores information pertaining to different words. The information comes from the feature extractor and is sent to the host via the serial link. Secondly, during the recognition phase, the Z-151 compares the information it has on each of the words in the library to the current test word in order to find a match. When a match occurs, the word is recognized. The concepts of a training phase and a recognition phase will be made clear in the next section.

2.3 Operational overview

Again, the voice recognition system is designed to respond to the voice of one person. The person operating the system has to "train" the system to his or her voice. Therefore, operationally, the recognition system has two distinct phases: a training phase and a recognition phase. The recognition phase is further divided into a library building part and a recognition part. The programs that run in the host computer during the training phase and the recognition phase are TRAINJFK and RMATCH5, respectively. Since RMATCH5 runs considerably slower

than TRAINJFK, RMATCH5 takes both its library words and the test words (defined later) from user defined disk files (more on this in Chapter 4). This enables the user to process large amounts of data (i.e., to recognize words) without having to be present. Also, it allows for consistent testing, as the same data can be used each time a change is made to the matching algorithm.

During the training phase, the input patterns (from the feature extractor) are associated with the desired text words (which are entered via the keyboard of the host computer) and stored onto a disk file. The disk file is known as the library disk file, and the words stored in the file are known as the library words. The library words are chosen by the user. Thus, during the training phase, TRAINJFK is used to create the library disk file.

During the library building part of the recognition phase, the TRAINJFK program is invoked several times so that comparison disk files can be created. Each time TRAINJFK is invoked, another comparison file is created. Each comparison file should have one copy of each library word in it. Thus, the collection of comparison disk files holds several utterances of each library word. These comparison files, along with the library file created during the training phase, are known as the library building files. Once the library building files are made, the RMATCH5 program is invoked. RMATCH5, when initialized properly (as described in Chapter 4), uses the comparison files to modify the library words and the library disk file. This ensures that a better library (one that will recognize better) will be built.

During the recognition part of the recognition phase,

TRAINJFK is again invoked several times, so that several test disk files can be created. Once the test files are made, RMATCH5 is invoked, using the appropriate initialization. RMATCH5 uses the built-up library (specified by the user), to recognize the words in the test files. The correct ways in which to invoke TRAINJFK and RMATCH5 are covered in Chapter 4.

CHAPTER 3

SYSTEM HARDWARE

The system hardware resides entirely in the feature extractor whose function was described in chapter 2. The system hardware occupies five nine-inch by four-inch vector circuit boards. One board, the microcontroller board, contains the 8751, the clock circuit, the external data buffer memory, and the A/D converter. Another board, the zero crossings detection board (ZC detection board), contains the zero crossing detectors for each frequency band and the microphone signal. The remaining three boards, the analog boards, contain the seven bandpass filters (BPFs) along with their peak detectors (PDs). One of the three analog boards also contains the microphone amplifier. The five boards are plugged into a common backplane through which they can interact with each other. The backplane provides a common ground for all the boards as well as the necessary power supply voltages. The exact pin definitions for the backplane can be found in Appendix C.

3.1 Analog boards

As mentioned previously, there are three analog boards on which are the peak detectors and the bandpass filters. The microphone amplifier, which is on one of the analog boards, takes the microphone signal from the backplane, amplifies it, and then puts the amplified signal back on the backplane. The amplified signal is taken off the backplane by each of the bandpass filters whose outputs go to the corresponding peak detector and to the backplane (for use by the zero crossing detectors). The output of each peak detector is also put onto the backplane.

The microphone amplifier consists of two stages, each stage being an operational amplifier circuit as shown in figure 6. The

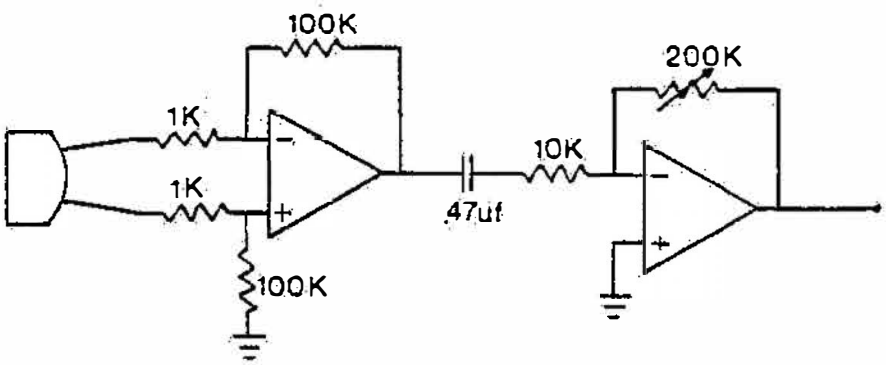


Figure 6 - Microphone amplifier.

first stage is a preamp circuit, designed for a dynamic microphone. The microphone used here is a General Electric dynamic microphone, commonly used with tape recorders. The

second stage of the microphone amplifier is an inverting amplifier gain stage, where the gain can be varied from 0 to 20. The gain of the last stage is adjusted so that the output of the microphone amplifier is no greater than +/- 5 volts. This ensures that the output of the peak detectors will not go above 5 volts, thereby avoiding the saturation of the A/D converter. (The saturation may arise since the A/D has a conversion range from 0 to 5 volts, while the operational amplifier circuits have outputs that can range from -12 to 12 volts.)

During the course of the thesis research, two sets of bandpass filters were tried. The first set consisted of four pole filters with designed center frequencies of 62.5 Hz, 125 Hz, 250 Hz, 500 Hz, 1000 Hz, 2000 Hz and 4000 Hz. The reason for the choice of center frequencies was discussed in section 2.1. Due to the unavailability of exact component values, the actual center frequencies were slightly different from the desired center frequencies. The center frequency shift is shown in figure 7, where the gain of each filter is plotted against frequency. From figure 7 it is evident that the frequency bands overlap significantly. Originally, the overlap was thought to be good for two reasons. First, if all the filter responses are added together, a uniform coverage of the speech frequency spectrum is obtained. Second, the low order filter design (i.e., four poles) needs relatively few components, thus reducing the total feature extractor circuitry. However, after some experimentation, it was clear that the filters did not provide enough frequency resolution. Energy coming into the system at 200 Hz would have strong influences in three bands: the 125 Hz

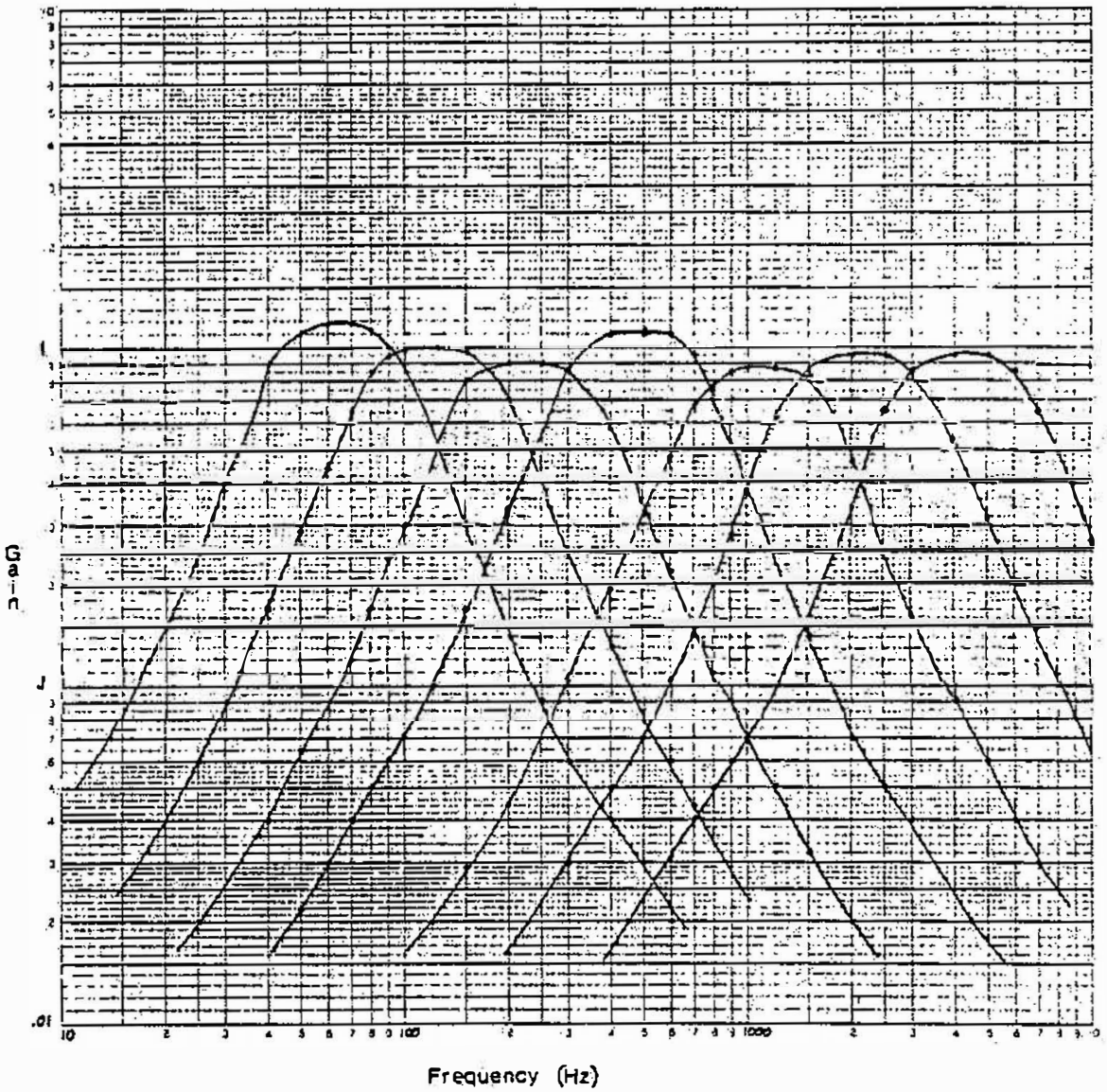


Figure 7 - Initial filters' frequency response.

band, the 250 Hz band and the 500 Hz band. The result is that the output of adjacent bandpass filters would look the same, thereby producing the same patterns. This renders half of the bands useless because they look the same as the other bands. Also, since energy coming in at a certain frequency will show up in two or three of the bands, it is difficult to determine the frequency content of the incoming signal. The inability to determine the signal's frequency content is inconsistent with the information that the feature extractor is supposed to supply to meet the design goals. Due to the inconsistency, a new set of higher order filters was constructed so that the gain responses do not overlap to such a great extent.

The higher order filters are ten pole or tenth order Butterworth bandpass filters, each filter having a gain stage at the output. The filter's Butterworth response is chosen because it is maximally flat in the pass band and also because this type of filter's design is relatively straightforward. The gain stage of the filter, shown in figure 8, is a non-inverting amplifier.

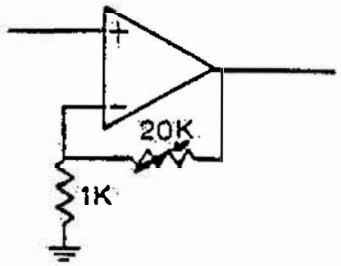


Figure 8 - Filter gain stage.

The amplifier design, because of its high input impedance, will not load down the output of the filter. Also, the addition of a

gain stage in each filter allows for more flexibility, since the gain in each band can be adjusted separately. The flexibility is useful because quiet bands can be given more gain so that a signal in that band can be detected. The gains were originally set so that the center frequency gain of each filter was as close to one as possible. However, after some experimentation, it was decided to set the gain of each band so that the detected signal (if there is a signal to be detected) would be in the range of 2 to 3 volts, under normal speaking conditions.

The choice of a tenth order filter is made based on a compromise. As mentioned above, it is desired to have as little overlap in the gain responses as possible. Thus, the order of the filter should be as high as possible (i.e., a high number of poles in the filter). On the other hand, it is also desired to keep the component count in the filter design relatively low, which advocates the use of low order filters. Thus, the tenth order filter is chosen for two reasons. First, a tenth order filter provides ample frequency separation, as shown in figure 9. Second, a tenth order filter is the highest order filter that will fit in the allotted space on the circuit board.

Using ten pole filters allows the designed crossover point in the gain response to be at 35 dB down. This amount of attenuation at the crossover point is more than adequate to provide the desired frequency separation. Figure 9 shows the gain responses of the new bandpass filters. The actual center frequencies of the filters are slightly shifted from the desired center frequencies due to the unavailability of exact component values. Since the center frequencies are somewhat off, it

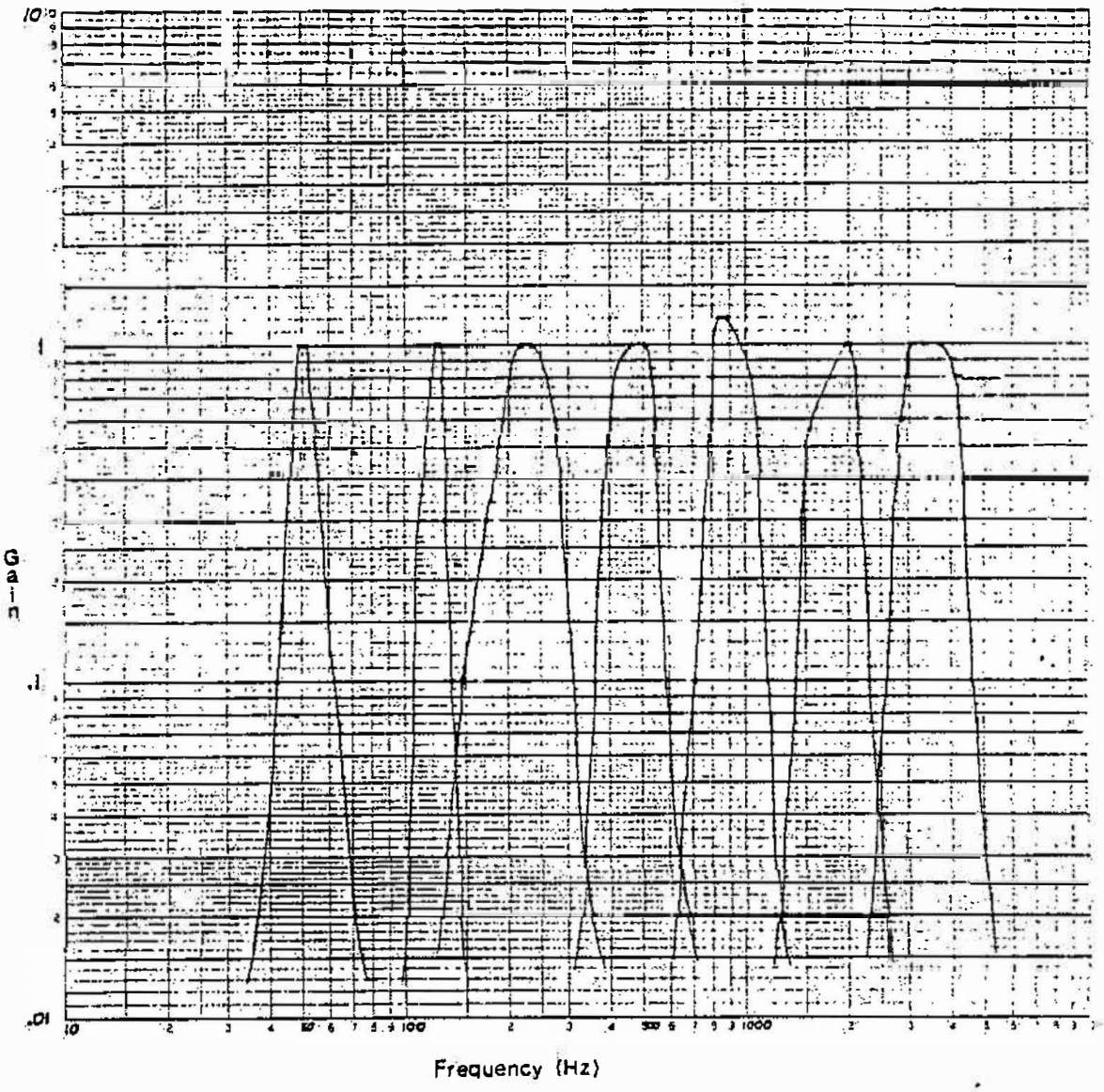


Figure 9 - Filters' frequency response.

follows that the gain crossover point will not always be at 35 db down; sometimes it is even lower or somewhat higher (see figure 9).

Each tenth order filter is composed of five identical (except for component values) stages. Each stage is an operational amplifier circuit, known as the Delyiannis-Friend circuit [8]. The circuit, shown in figure 10, is chosen because of its good stability and low component count. The component values for each stage of each filter are given in Appendix A [8].

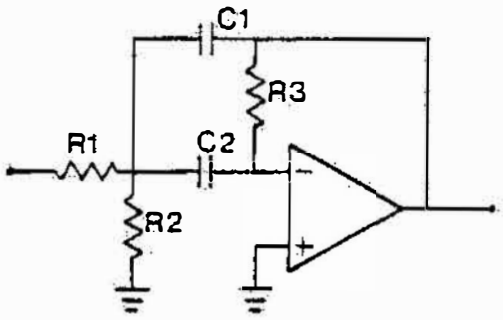


Figure 10 - Bandpass filter stage.

The output from each bandpass filter is an AC waveform corresponding to part of the speech signal which resides in a particular frequency band. However, the recognition system needs only the envelope of the waveform, so that an overall pattern can be detected. The envelope corresponds to a peak detection of the AC signal. Rather than accomplishing the peak detection in software, it has been decided to use hardware.

The first peak detector circuit that comes to mind is the diode-capacitor peak detector shown in figure 11. However, since

speech signals tend to be asymmetric with respect to zero volts, the simple half-wave rectifier circuit is not appropriate.

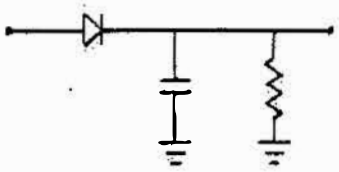


Figure 11 - Diode-capacitor peak detector.

Immediately, a full-wave diode rectifier circuit comes to mind. However, the full-wave rectifier also suffers from a major problem, namely, the constant 1.4 voltage drop accross the diodes. Since the speech signal is many times under 1.4 volts, this circuit is also unacceptable. In order to overcome the diode voltage drop problem, a full-wave operational amplifier peak detection circuit is chosen. The design of the circuit is shown in figure 12 [9].

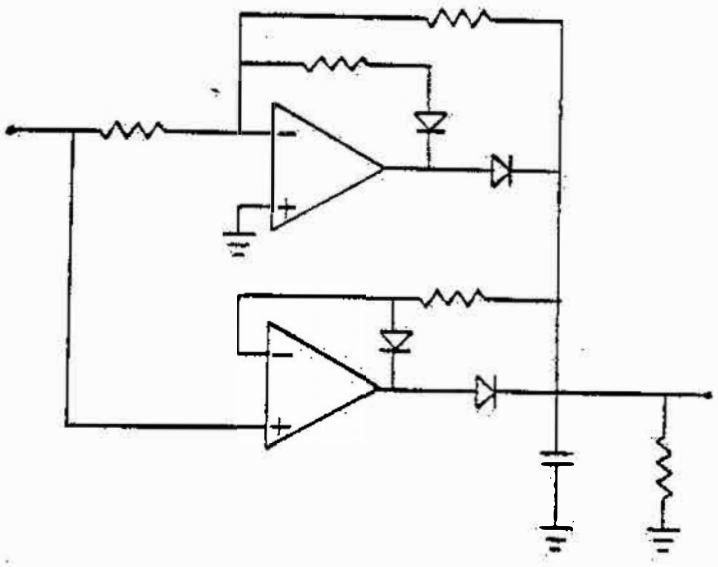


Figure 12 - Operational amplifier peak detector.

Upon inspection of the circuit in the figure, it is seen that it is a full-wave rectifier with a capacitor and resistor on its output going to ground (forming an RC time constant). Thus, the peak detected signal will strictly follow the input waveform on its positive slope while on the negative slopes it will fall at a rate determined by the RC time constant. For a large RC time constant, the input and output waveforms are shown in figure 13a. The output is obviously not a good representation of the

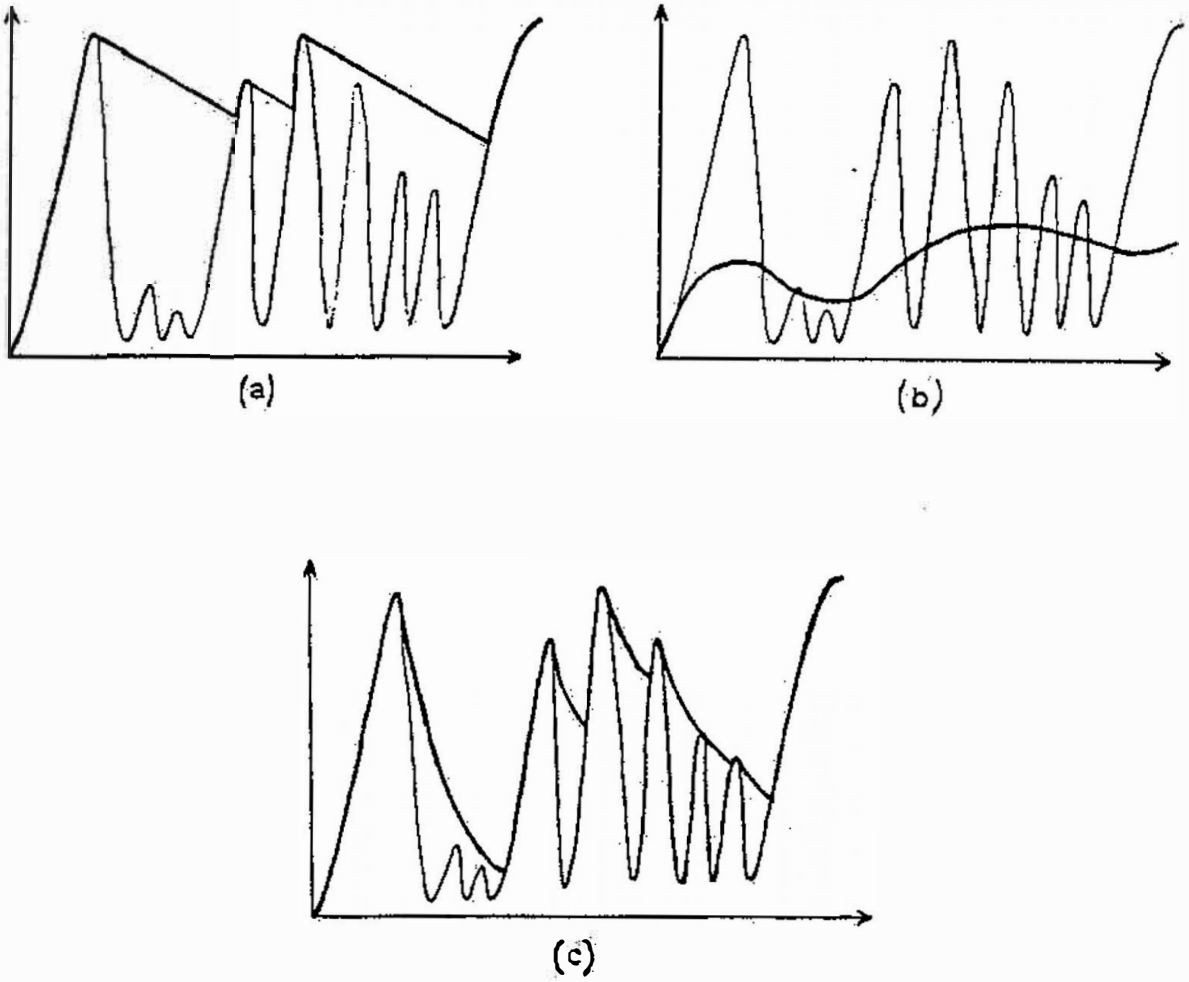


Figure 13 - Peak detection.

input waveform's envelope. One problem is that the RC time constant is too large. Another problem, not immediately obvious, is that a voltage spike on the input will cause the output to move away from the waveform envelope. Thus, for high amplitude spikes on the input, it is desirable to keep the output from following the input. This is accomplished by the addition of a series resistor inserted before the RC circuit, as shown in figure 14. The figure shows the peak detector as used in this

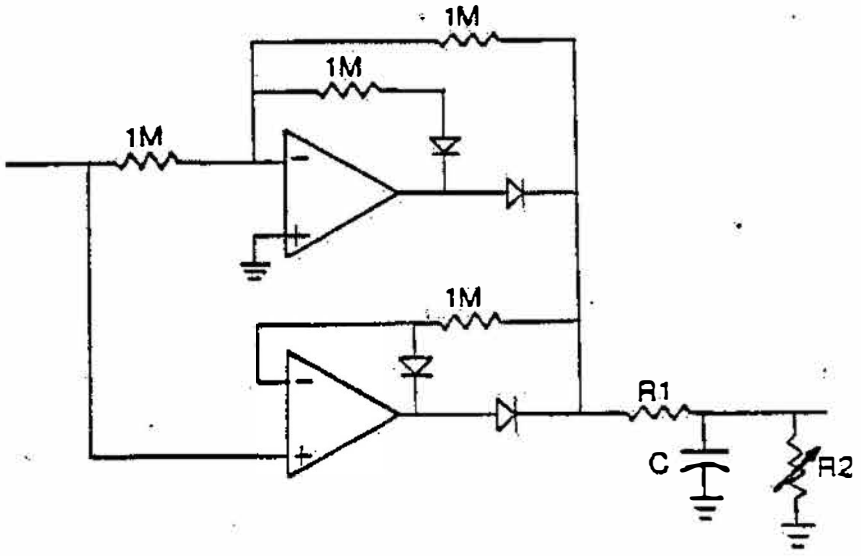


Figure 14 - Modified peak detector.

thesis. Referring to figure 14, if R1 and R2 are chosen too large, then the output waveform does not follow the input waveform closely at all, as shown in figure 13b. The desired behaviour of the peak detector circuit is shown in figure 13c. Therefore, the values of R1 and R2 are chosen empirically to achieve the desired behaviour, keeping in mind that R1 should be as large as possible and R2 as small as possible. Values for R1

in all the peak detectors are given in Appendix A.

3.2 Zero crossing detector board

As mentioned previously, each band has a zero crossing detector (or ZC detector). The circuit for each detector of each band is the same (see figure 15). The inputs to the ZC detectors are taken from the backplane (the signals originate on the analog boards). The output of each detector is connected on the backplane, to a common data bus (known as the data bus), that is also connected to port 1 of the 8751 microcontroller. The 8751 resides on the microcontroller board (to be discussed below).

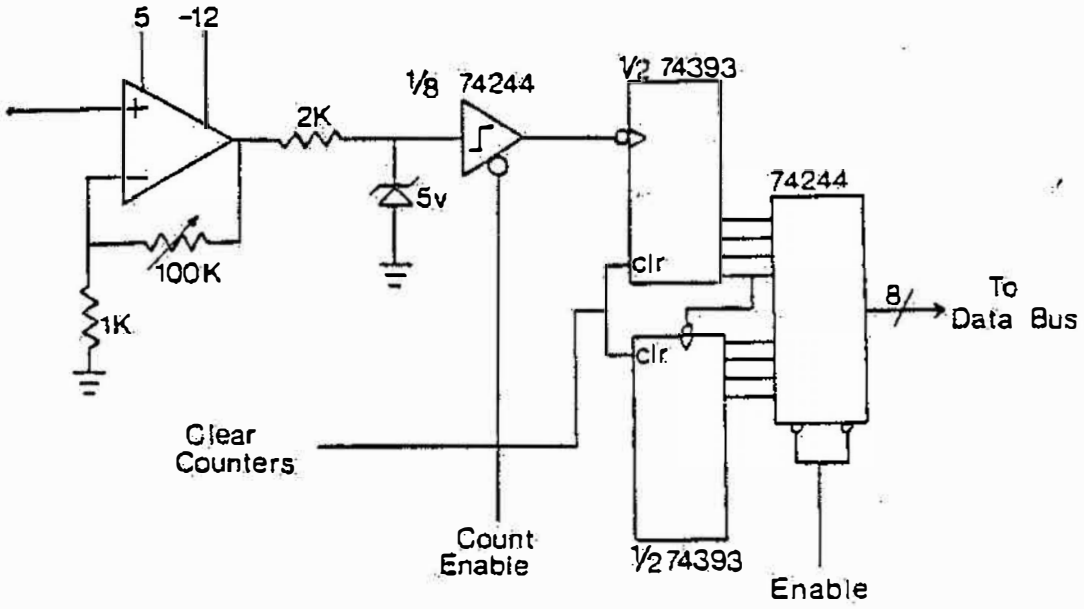


Figure 15 - Zero crossing detector.

The general operation of each detector is as follows: The incoming signal is amplified so that it clips at both the positive and negative supply voltage values (of the amplifier). The negative portion of the signal is discarded, and the positive portion of the signal is clipped at +5 volts (a zener diode performs both functions, see figure 15). The resulting signal is used to clock a counter which keeps track of the number of zero crossings that occur within a given time interval. Upon close inspection of the above description, it is evident that the detector does not count every zero crossing; rather, it counts every other zero crossing. Thus, these circuits would be more appropriately called cycle detectors. However, the name zero crossing detectors will be used.

The first stage of the ZC detector, as shown in figure 15, is a non-inverting operational amplifier gain stage. The design is chosen for its high input impedance so that the input signal is not loaded down. The gain of the stage can be adjusted from 1 to 101. The gain is adjusted so that an incoming signal that is smaller than 0.05 volts, zero to peak, will not trigger the counter.

The supply voltages of the operational amplifier are split for several reasons. Normally, it would operate on +/- 12 volts (as they do for the bandpass filters). However, since the output of the operational amplifier will be kept below 5 volts, there is no reason to amplify the signal beyond this point. Thus, the positive supply voltage is set at + 5 volts. Following the same reasoning, since the negative half of the signal will be discarded, it follows that the negative supply voltage should be

grounded. This would be a good solution, except that the operational amplifier's performance is severely degraded when a single supply is used. Since the only negative voltage available in the system is -12 volts, the negative supply voltage is set to this value. The particular operational amplifier chosen for the circuit is the Texas Instrument's TL074CN. The package is chosen for its superior frequency response and because it is a quad-operational amplifier package (helping to reduce the number of components).

After the first stage of the ZC detector, the signal has been clipped at +5 volts and -12 volts. The zener diode, as shown in figure 15, transforms this signal into a TTL signal (0 volts to 5 volts). It ensures that the signal never exceeds 5 volts, and it grounds out the negative portion of the signal. The 2 kilohm resistor is included to limit the current when this "grounding out" occurs. After the zener diode, a 1/8 74244 Schmitt triggered buffer is inserted for two reasons. First, it squares up the signal before it goes to the counter (the 74393). Second, it allows the ZC detector to be effectively enabled or disabled; (the counters must be disabled or stopped before their contents can be examined). Since a 74244 package contains 8 buffers, and since there are 8 detectors, it follows that one 74244 package is used to buffer all the signals before they go to their respective counters. In this configuration, all the counters can be enabled or disabled by controlling the logic level on one line.

After the buffer, the signal is used to clock a 74393, which is configured as an 8 bit counter, as shown in figure 15. The

output of the counter is connected to a 74244 buffer. The purpose of the buffer is to effectively connect or disconnect the output of the counter to the data bus.

Figure 15 shows three control lines, CLEAR COUNTERS, COUNT ENABLE, and ENABLE. The first two are common control lines, meaning that one of each control line goes to all the ZC detectors. Thus, by controlling one line, all the counters can be enabled or disabled. By controlling another single line, all the counters can be cleared. In contrast, there are eight ENABLE lines, one line for each ZC detector. Figure 16 shows the circuit that sends these control signals to the detectors.

The control circuit for the ZC detectors, as shown in figure 16, is simply a memory mapped latch. The 8751 (which resides on the microcontroller board) can access 64 kilobytes of external data memory. Since only 8 kilobytes are used, any address outside the range 2000H to 3FFFH (to be discussed in section

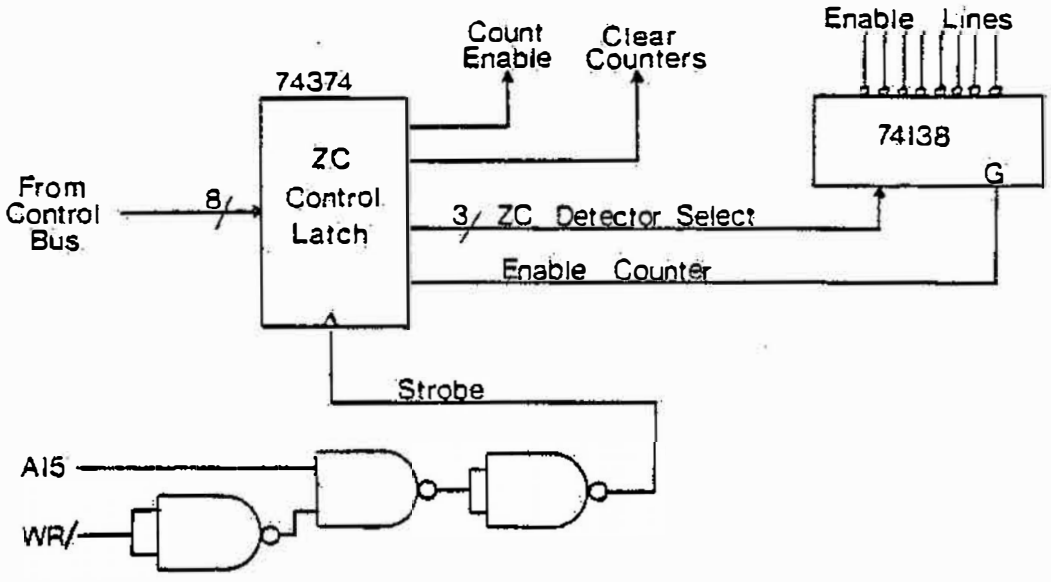


Figure 16 - Zero crossing control circuit.

3.3) is acceptable for the ZC control latch address. Here, all addresses between 8000H and FFFFH are chosen as the latch address. The choice makes the address decoding logic simple, as only the highest address line (from the 8751), A15, needs to be monitored. In addition to A15, the WR/ signal from the 8751 must also be monitored. When WR/ goes low, it indicates that there is a valid byte on the 8751's port 0 (which is connected to the control bus on the backplane). Thus, each byte is strobed into the ZC control latch only when A15 is high and WR/ is low, see figure 16.

By writing different bytes to the ZC control latch, via the control bus (discussed above), the counters can be cleared, enabled (to count) or disabled (to stop). Also, any one of the counter's contents can be put onto the data bus, connected to port 1 as discussed above, so that it may be read by the 8751.

3.3 Microcontroller board

As mentioned previously, the microcontroller board contains the 8751, the A/D converter, the clock circuit, and the buffer memory. The microcontroller board is designed around the Intel 8751 microprocessor chip. The integrated circuit (IC) has 128 bytes of on chip RAM, 4 kilobytes of on chip EPROM, one serial I/O port, two programmable timers, and up to 32 I/O port pins. The 8751 is designed for control applications, with numerous bit oriented instructions with which to manipulate the port pins [10].

In the present work, some of the I/O pins are used to

control the A/D converter, some are used to connect to the data bus, while others are used as a serial port for communicating with the host computer. Due to the flexibility of this IC, the I/O interfacing is very simple, as will be described later.

The function of the board is to convert the outputs of the peak detectors in each band into digital form, to gather the data, and to collect the data from the ZC detectors. All the data are then encoded and sent to the host computer for further processing. The signals coming into the board (from the backplane) are the outputs of the seven peak detectors (which go to the A/D converter), the data bus and a serial line. The outgoing signals from the board are the control bus and a serial line. As mentioned before, the data bus is an eight-line wide bus that is used to bring data to the 8751. The control bus is also an eight-line wide bus that is used to control the ZC detectors. The data bus is connected to port 1 of the 8751 while the control bus is connected to port 0 of the 8751.

Since the 8751 has to buffer the input data before sending it to the host computer, and since the 128 bytes of on chip RAM are not enough to serve the purpose, additional RAM IC's are used in the design. With a frame (defined in section 2.1) being collected every 10 ms, each frame being 31 bytes long, 8 kilobytes of RAM can hold 2.6 seconds of speech, which is appropriate for our purpose. Therefore, an 8 kilobyte RAM is added to the microcontroller board.

The 8751 uses multiplexed address and data lines; thus, the lower 8 bits of the address share the same pins as the data lines. Because of the multiplexed lines, the RAM IC chosen for

the design is the Intel 8185, which is especially designed for this type of processor. The 8185 contains 1 kilobyte of RAM, organized as 1024 by 8 bits, with a built-in latch for the multiplexed address lines [11]. Thus, the 8 kilobytes of RAM desired are attained with eight 8185 IC's.

The address decoding circuitry for the RAM uses the upper 3 address bits to enable the chip select circuit, which decodes the next three address bits to enable one of the eight RAM chips. The remaining 10 address bits are passed to each of the RAM chips (but only the selected chip uses them). A single IC, the 74LS138, is used to perform the above decoding function. As shown in figure 17, the decoder accepts an address range of 2000H to 3FFFH, enabling one of the eight 8185 RAM IC's for each one kilobyte range.

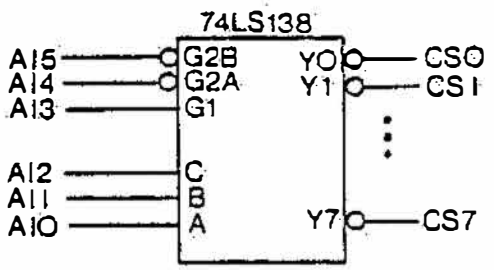


Figure 17 - Address decoder.

As mentioned before, the 8751 contains an on chip serial port. The port derives its serial transmission rate from the clock that drives the 8751. Thus, the exact clock frequency for the 8751 is influenced by the desired serial transmission rate. It is decided that the serial port is to transmit at as high a

rate as the host computer can handle. This is determined to be 9600 bps (bits per second) for the Zenith Z-151.

The 8751 clock frequency is determined by the following equation:

$$f = r(384\ (256-x))$$

where r is the desired bit rate and x is a one-byte value that is loaded into timer 1. Other constraints are that f be less than 8 MHz for this version of the 8751, and that x must be an integer. Given the above constraints, it is decided to let x have a value of 254. With r equal to 9600 bps, the clock frequency for the 8751 needs to be 7.3728 MHz. How the clock frequency is generated will be discussed later.

Since the on chip serial I/O is done at TTL levels, some level shiftings must be done so that the serial I/O will be at RS232C standard levels for communicating with the host computer. Standard line driver and receiver IC's, the MC1488 and the MC1489, respectively, are used to accomplish the level shiftings. The serial lines are brought out of the microcontroller board through the backplane, to a DB25 connector. From here, a cable connects the serial lines to the host computer.

The Intel 8751 also has a built in reset on power up circuit. Although Intel recommends using an external 10 microfarad capacitor [10], this does not give a reliable reset pulse. Therefore, a 33 microfarad capacitor is used. With this value, the actual measured reset pulse duration is more than one second.

As mentioned before, the microcontroller board contains an

A/D converter, which is used to digitize the outputs of each of the peak detectors. The A/D used is an ADC0816 single chip 8 bit converter. This chip has a built-in 16 channel analog multiplexer, with 4 address input lines. With a clock frequency of 640 KHz, the ADC0816 has a typical conversion time of 100 microseconds. The conversion is done using the successive approximation method, with all the operations performed on the chip. Thus, to initiate a conversion, only a start signal needs to be sent to the IC. When the conversion is complete, the IC signals this by asserting an end-of-conversion (EOC) signal. In addition, the digital output is through an 8 bit register with tri-state capability, allowing direct connection to the data bus (on the backplane) [12]. This feature is particularly useful since the data bus can be made common to all the ZC detectors, the data output of the A/D, and the address input to the A/D (to be described next).

As indicated above, the 4 address lines and the 8 data lines of the ADC0816 share the same 8 port pins (port 1) of the 8751. This is possible since when the conversion is in progress, the 8751 can latch the proper address into the ADC0816, using the ALE input of the converter IC. At the same time, it inhibits the output from the A/D, by sending a logic zero to the OUTPUT ENABLE pin. When the conversion is completed (signal EOC), the 8751 switches its port pins to input mode, and then enables the output of the ADC0816, while disabling everything else on the data bus (i.e., the ZC detectors). Thus, the data flow from the ADC0816 to the 8751 can be accomplished.

Because the ALE and START signals of the ADC0816 are

triggered on the leading and falling edge, respectively, they are tied together and are driven by a single signal from the 8751 (P3.3). This helps to reduce the usage of the port pins. In addition, the EOC signal from the converter chip is sent to P3.2 of the 8751. This allows the 8751 to sense the end of conversion, rather than waiting for the maximum conversion time, thus allowing a more efficient use of the CPU.

To determine the clock frequency of the ADC0816, a few constraints must be observed. One is that, according to the specifications, its permissible clock range is from 10 KHz to 1.28 MHz. Another is that the clock frequency must be easily derivable from the system clock. Since a maximum conversion time of 120 microseconds (74 clock cycles) is appropriate for our goals, the clock frequency for the ADC0816 has been made 614.4KHz.

3.4 Clock signals

As mentioned above, the clock frequency for the 8751 is 7.3728 MHz. Unfortunately, a crystal at this frequency is unavailable. Thus, to generate the desired clock frequency requires a crystal whose frequency is some integer multiple of 7.3728 MHz. The lowest frequency where a crystal is available is 22.1184 MHz, three times the desired frequency. Therefore, a crystal of this value is used in the primary clock (from which the 8751 clock and the ADC0816 clock are derived).

A 74LS629 IC is used as the primary clock circuit. It should be noted that, although the clock circuit as designed

works well, the 74LS629 is rated only to operate at a maximum frequency of 20 MHz.

To drive the 8751, this main clock frequency of 22.1184 MHz needs to be divided by a factor of three. This is accomplished by a 74LS92 IC, thus giving the 7.3728 MHz clock for the 8751. To drive the ADC0816, the clock for the 8715 is further divided by 12, using another 74LS92 IC. This gives the seemingly odd frequency of 614.4 KHz for the ADC0816 clock. The complete clock circuit, as described above, is shown in figure 18.

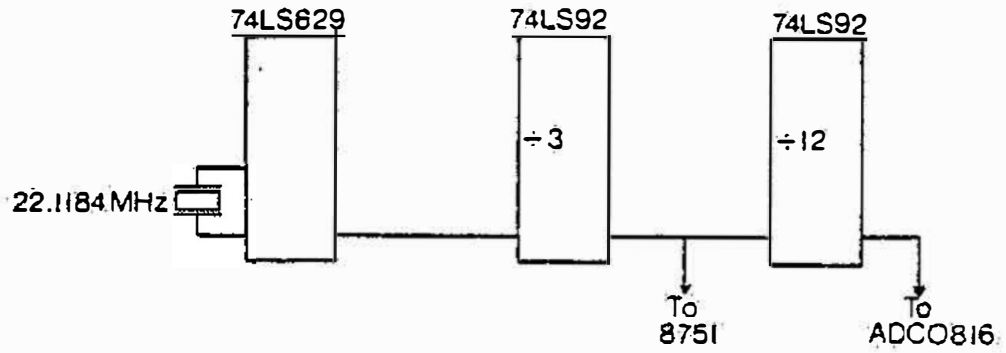


Figure 18 - Clock circuit.

CHAPTER 4

SYSTEM SOFTWARE

Three programs are essential to the operation of the voice recognition system. Within the feature extractor, the 8751 runs an assembly language program called LIS. It is common to both the training and recognition phase. In the host computer, the turbo pascal programs TRAINJFK and RMATCH5, are used for the training phase and the recognition phase, respectively. Each of the three programs will be discussed at length in the text that follows. Listings for each program can be found at the end of the thesis.

4.1 LIS

As mentioned before, LIS is the program that resides within the 8751. It is responsible for sending signals to the hardware

of the feature extractor, in order to control the sampling and collection of data. LIS manages a circular data buffer, and also communicates with the host computer. In addition, it also gives the feature extractor a certain amount of intelligence, making it able to detect the word boundaries.

Figure 19 shows the main loop of LIS in flowchart form. It indicates the sequence of events which takes place during the sampling and collection of data. To enter the main loop, the host computer sends a command (STCMD) to the 8751, to tell it to start looking for a word. This state of the program (i.e., when LIS is looking for a word), is indicated by setting a flag (ST). After LIS has determined that it should start looking for a word, a timer is set up so that a consistent system sampling period is obtained, see figure 19.

The sampling period is timed using timer 0 of the 8751. The timer is set up at the start of a sampling period that expires 10 milliseconds later. LIS then determines the timer's status by examining bit TF0, which is set when the desired amount of time has elapsed. The 10 milliseconds is the system sampling period. The system sampling period is formally defined as the inverse of the rate at which frames of data (defined in section 2.1) are collected. During the course of the thesis, 10 milliseconds was found to be a good system sampling period because it is the longest period that can be used in which the desired patterns are readily observable. A higher value than 10 milliseconds, corresponding to a slower sampling rate, is too slow to pick up the desired patterns from the peak detectors. A lower value than 10 milliseconds, corresponding to a faster sampling rate, will

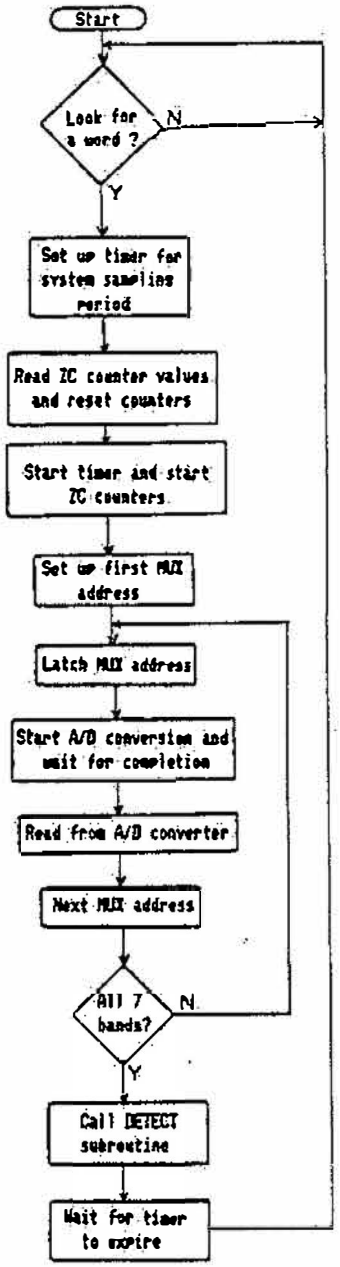


Figure 19 - Main loop of LIS.

give more frames per word, but no significant additional information.

Returning to figure 19, after the timer is set up, the ZC counters are read into a temporary buffer and then reset. The task is accomplished using a subroutine called CNTR (see LIS program listing). Since the counters are initially set to zero and disabled, it follows that the ZC values in the first frame will all be zero. However, due to the nature of the system, it is rare that this first frame of data is included as part of a word, because by the time the host prompts the user to say the word, the first frame of data has already gone by.

After the ZC counters are read, they are started along with the timer, as indicated in figure 19. The first multiplexor address for the A/D converter (MUX address) is then set up in preparation for the inner loop. The inner loop scans each of the seven bands, converting the peak detector amplitudes into digital data, and storing them into the temporary buffer. The scanning takes place as quickly as possible, given that the program steps can not be executed instantaneously. Note that in the code of the main loop, there exists a settling time in which the 8751 waits. The settling time occurs after the latching of the multiplexor but before the starting of A/D conversion. During the course of the thesis, a log amplifier was connected between the output of the multiplexor and the input of the A/D converter. This amplifier was to help normalize the incoming signal and to make the system insensitive to the speaker's volume level. The settling time was included so that the log amplifier would have time to settle down before the A/D conversion began. However,

because of nonlinearities and noise problems, the log amplifier was disconnected (although the circuit still remains on the board), and the settling time is minimized.

After the inner loop is completed, a subroutine called DETECT is invoked. The subroutine takes the data from the temporary buffer, converts it into Ascii hex digits, and then stores it in the external data buffer. The forementioned is accomplished through a call to subroutine CHECK (other subroutines involved are SEND, CONVERT, and PUTIN, see program listing). DETECT also gives the feature extractor its intelligence, because the subroutine enables it to detect word boundaries, thus the name DETECT. In addition, DETECT keeps track of the state of the program through the use of various flags (see LIS program listing). There are four states in which the program can be in. LIS can be looking for a word, but has not found one yet; or it may have found a word and is currently taking in more data pertaining to that word. A third state is if the end of a word has been detected and there are still data in the external buffer to be transmitted to the host computer. The fourth state is when the end of the word has been detected and there are no more data in the external data buffer to transmit (i.e., an idle state or a starting state).

The word boundary detection algorithm used by DETECT is based on the output of the peak detectors (i.e., the ZC counters are not used for word boundary detection). Each of the seven bands has a threshold associated with it. THR1 is the threshold for the 62.5 Hz band up to THR7 for the 4000 Hz band. These thresholds are variables, which are set to default values when

the program is initialized. The beginning of a word is declared when at least one of the peak detector outputs goes above its respective threshold for a variable number of consecutive frames. The variable is called TSAM (see LIS program listing). The beginning of the word is said to be the first frame in the consecutive string of frames. Note that the above condition does not mean that the same peak detector output must be above its respective threshold for #TSAM consecutive frames. An example will illustrate the point. Suppose TSAM is equal to three and the first frame of data collected (while looking for a word) shows the 500 Hz band (i.e., the output of the peak detector attached to the 500 Hz bandpass filter) is above its threshold. The next frame of data shows the 1000 Hz band is above its threshold. If the third frame has any one of the bands above its respective threshold, then the beginning of a word will be declared and the first frame of data taken becomes the first frame of data for the word. The end of a word is declared after all the bands go below their respective thresholds for a variable number of consecutive frames. This variable is called SPCC (see LIS program listing). The end of the word is said to be the last frame in the consecutive string of frames.

When DETECT is called while LIS is collecting data, a first in, first out circular buffer management scheme is used for the external data buffer. For this, two pointers are needed, a "put" pointer, which points to the next available memory location, and a "get" pointer, which points to the next byte to be taken out of the buffer. When the "put" pointer reaches the end of the external memory space, it wraps around to the beginning. Thus,

as data are taken out of the buffer and sent to the host, the room is made available for incoming data. The scheme ensures that the buffer is utilized to its maximum potential. The buffer is full when the two pointers indicate the same location after a byte has been put into the buffer. The buffer is empty when the two pointers indicate the same location after a byte has been taken out of the buffer. The two subroutines responsible for the buffer management are PUTIN and GETOUT. PUTIN puts a byte into the buffer while GETOUT takes a byte out.

When the DETECT subroutine is called while LIS is looking for a word, a unique buffer management scheme is employed. Each time the DETECT subroutine is called and there is at least one band above its threshold, the current frame of data is taken from the temporary buffer, converted to ascii hex format, and put into the external data buffer (which is accomplished through a call to subroutine CHECK). If at any time before the beginning of a word is declared, a frame of data is obtained that has all the bands below their thresholds, a unique action is taken: the external data buffer's pointers are reset. The action effectively wipes out the information in the external buffer. Subroutine CHECK performs the resetting of the buffer's pointers. The deletion of the data in the external buffer is necessary because once the consecutive pattern is broken, there is no longer any possibility that the data can be part of a word. If the beginning of a word is declared, then all the frames of data pertaining to that word (i.e., the frames of data that led to the word being declared) are saved in the external buffer. The unique scheme is used so that no frames of data that pertain to a

word are lost.

As mentioned before, the data are converted to an ascii hexadecimal format before being put into the external data buffer. Thus, each 8 bit data byte is encoded into 2 ascii hexadecimal characters, with the high order character being put into the lower memory location of the external buffer. Thus, each frame of data is represented by 30 ascii bytes in the external data buffer instead of 15 bytes, as it is represented in the temporary buffer (8 ZC counters and 7 PDs). The encoding is done for two reasons. One, most high level languages can not handle pure binary data. Secondly, if displayable codes are used, the program can be executed using a terminal as a host.

The format in which the ascii data are stored in the external buffer will now be presented. The organization of each frame, from lowest memory location to the highest memory location, is: PD(62.5Hz), PD(125Hz), PD(250Hz), PD(500Hz), PD(1000Hz), PD(2000Hz), PD(4000Hz), ZC(Microphone), ZC(4000Hz), ZC(2000Hz), ZC(1000Hz), ZC(500Hz), ZC(250Hz), ZC(125Hz), ZC(62.5Hz). Here, PD(125Hz) means the output of the peak detector of the 125 Hz band, ZC(500Hz) means the ZC count in the 500 Hz band, and so on. Also, when the ascii digits are stored in the external buffer, a NL (newline) character is inserted after each frame of data to mark the end of the frame. When the end of a word is detected, an EOWM, NL sequence of characters is inserted into the external buffer, marking the end of the word (see program listing for definition of EOWM character). If LIS is not commanded to take data after the end of the word is detected, then an EOF, NL character sequence is inserted into the

data buffer immediately following the two characters previously mentioned (see program listing for definition of EOF character). This character sequence marks the end of all usable data that are in the external buffer (i.e., an end of file marker). If LIS is commanded to take data after the end of the word, then these frames of data would be in between the EOWM, NL character sequence and the EOF, NL character sequence.

As mentioned before, LIS also communicates with the host computer. The communication is command oriented, meaning the host issues a command and LIS responds. There are 14 ascii characters corresponding to 14 commands that LIS will accept. Most of the commands were created for experimental purposes; thus, only the most important ones will be covered (later in this chapter).

Almost all the communication with the host computer involves the use of the interrupt service routine (the one exception is when the transmission of an ascii data byte is initiated by the subroutine PUTIN when it calls subroutine MAYBE, see program listing). A flowchart of the interrupt service routine is shown in figure 20. There are two ways to enter the interrupt service routine. First, when the serial transmission of a data byte is complete, the 8751 generates a serial interrupt, which automatically forces a call to the interrupt service routine. Secondly, when a valid data byte is received at the serial port (e.g., a command byte), the 8751 again generates a serial interrupt, which automatically forces a call to the interrupt service routine. In both cases a serial interrupt is generated. It is up to the service routine to determine whether

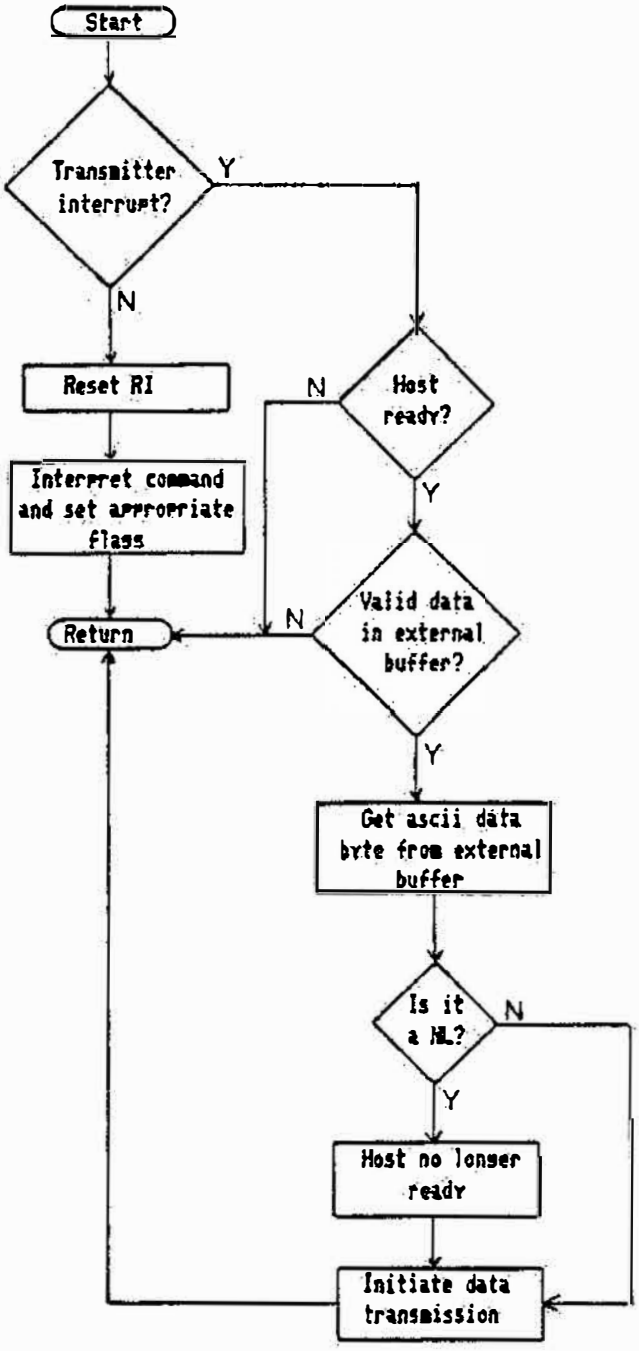


Figure 20 - Interrupt service routine.

the serial interrupt was generated by the receiver or the transmitter, as shown in figure 20.

When the host computer is ready to accept a word input, it sends a STCMD command to LIS which tells it to start looking for a word. Then, for each frame of data, the host requests the frame by sending a RQCMD command, at which time LIS sends the frame of data (30 ascii bytes) and the NL character (which marks the end of the frame). The host continues to request frames of data until the end of the file, when an EOF, NL character sequence is sent.

Upon the reception of each command byte, the LIS program execution branches to the interrupt service routine. Here it is determined that the serial interrupt is caused by the receiver, as shown in figure 20. The command byte is interpreted, and the appropriate flags are set, so that upon return to the main program, the desired actions will be taken. Note that each time a frame of data is requested, the interrupt service routine will be called 31 times. This is because after each byte is transmitted, a serial interrupt occurs, indicating another transmission may be initiated. On the 31st byte, when an NL character is being sent, the interrupt routine will reset a flag, indicating that the host is no longer ready to receive information, see figure 20. The reset flag will cause the LIS program to stop sending data until it receives another RQCMD command.

Up to this point, two commands, which LIS understands, have been discussed. The first is the STCMD command (which tells LIS to start looking for a word), and the second is the RQCMD command

(which requests a frame of data). The two commands are the only ones needed for the normal operation of the voice recognition system. However, a few other important commands will be discussed.

The ACMD command activates an averaging function. The averaging, which is performed by subroutine SEND, takes place just before the data are encoded and stored in the external data buffer. Since the averaging is done immediately before encoding, it does not affect the word boundary detection (i.e., the detection still uses the unaveraged frames). In preparation for the averaging, two temporary buffers are used. One is located at TBNEW, and it holds the current frame just collected (see LIS program listing). The other is located at TBOLD, and it holds the previous frame, or the one collected just before the current frame. Each time a new frame is collected, the frame at TBNEW is moved to TBOLD, and the new frame is stored at TBNEW (the frame that was in TBOLD is lost). The averaging function takes the frame at TBNEW and averages it with the frame at TBOLD. The averaged frame is the one that is encoded and stored in the external buffer. When two frames are averaged, it indicates that each byte of the current frame is averaged with the corresponding byte of the previous frame. Initially, the values stored at TBOLD are all zeros. Thus, the first averaged frame will always be the first collected frame divided by two. This causes no problems since the first frame is rarely included as part of a word (as discussed earlier). The DACMD command turns off the averaging function.

The TSCMD command tells LIS to collect 5 frames of data

after the end of a word has been detected. The task is accomplished through the use of a second main loop, which executes 5 times (see LIS program listing). As mentioned before, these 5 frames of data are inserted into the external data buffer between the EOWM, NL and EOF, NL character sequences. The command is included so that the end of the word detection algorithm can be evaluated. During the course of the thesis, it was observed that the inclusion of the frames (mentioned above) did not improve the system performance. Thus, the end of the word detection algorithm works well. The NTSCMD command tells LIS not to collect any more frames of data after the end of a word has been detected (see LIS program listing).

The PCMD command tells LIS to enter a parameter subroutine. Upon entering this subroutine, the following parameters are sent out from the serial port in ascii hexadecimal format: SPER, STLTM, THR1, THR2, THR3, THR4, THR5, THR6, THR7, SPCC and TSAM. SPER is the sampling period value, which determines the system sample rate. STLTM is the log amplifier settling time, which should always be set to the lowest value, 01 (see LIS program listing). THR1 through THR7 are the seven thresholds for the seven peak detectors. SPCC and TSAM are variables used in the word boundary detection (this was discussed earlier). Once all these values have been sent out, the subroutine waits to receive 24 ascii bytes, which will be converted to binary data and used as the new values for the above parameters. All the parameters are two ascii bytes except for SPER, which is four ascii bytes. Through the use of the command, the best values for the parameters can be obtained. As mentioned previously, a good

value for SPER is one that makes the system sample period equal to 10 milliseconds. The optimal values for the other parameters will be presented in Chapter 5.

The BCMD command (cold boot command) tells LIS to boot itself. This initializes the external buffer pointers, the temporary buffer and the program flags; it sets all the program parameters back to their default values and then restarts the program from the beginning. The WBCMD command (warm boot command) is exactly like the BCMD command except that the program parameters are not set to their default values.

Upon inspection of the LIS program listing, it may be doubtful that 10 milliseconds is enough time to collect a frame of data. However, this is not the case. Figure 21 shows how

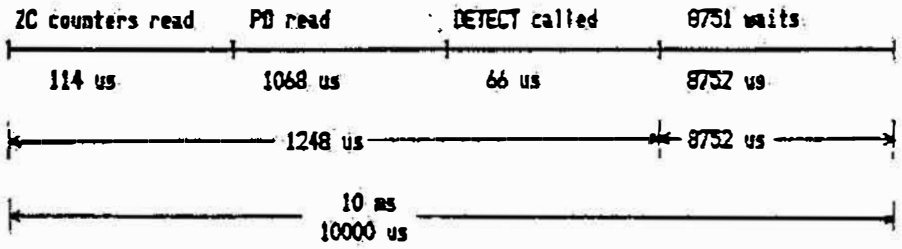


Figure 21 - LIS timing diagram.

much time is used by each operation in the main loop of LIS. The time durations are based on a clock frequency of 7.3728 MHz and the assumption that serial interrupts do not occur (i.e., the serial interrupts are neglected). Also, since the length of time required by DETECT depends on the program state, a worst case estimate is used there. Figure 21 shows that the data collection takes a total of 1248 microseconds, not even one-fifth the total sample period. Thus, 10 milliseconds is ample time to collect a

frame of data.

4.2 TRAINJFK

As mentioned before, the host runs the turbo pascal program TRAINJFK during the training phase of the system. When the program is invoked, it first prompts for the name of the new vocabulary file; thereby allowing the user to name the disk file where the data are stored. Thus, by invoking the program several times, the library file, the comparison files and the test files can be made. Next, TRAINJFK asks for the minimum frame count, which is the minimum number of frames a word can have. The value should be chosen large enough to reject partial words but small enough not to ignore a full word. A value of 5 is a good compromise. The third question asks for the number of frames to chop off the end of a word. The program uses the value entered (a decimal number) to truncate the last few frames which correspond to silence. The value should agree with the SPCC value of the LIS program (a good value for SPCC is given in Chapter 5).

The vocabulary file, whose name is defined by the user, essentially contains the raw data from the feature extractor. The file is made up of records representing each word in the vocabulary. Each record contains the text word itself, the encoded data frames, an end of word marker and an end of file marker. A sample record for the word NORTH is shown in figure 22. Here the newline separators between the frames were replaced with a carriage return linefeed for the purpose of illustration.

```
NORTH
01010E020001010100020103020001
01021C040001010100●50205030101
01031E040101020100060305020201
01032304010102010006030503020l
01032305010102010207040603010l
0102230D020203020A0D060602010l
0103241C050206021 60F080603010l
0105231D0F020803170B0306030101
01072411 1A020704160B0A05030201
01082412 1C010504140B0B05030201
0109251C1D0003041 0090B05020101
01092726 1D020304120 90B06030101
0109242919030405150B0A06030101
010A22211A03●40615●E0906020101
01091F192A020405151 00906030101
01081D1E3D010405150E090603●101
01071B2A42020406170C0A06020101
0106193033020304●19110A04020101
0106172C22020305161 40A06020101
0105122419030205141 40A06020201
01030F1B100202051213 0B06020201
0102101 70B0302050F130A06020101
0102101707030204101309060301 01
01010D17050302030F1 10907030101
01010B1B030302030A1 10A07020101
01010C1F030402020A130A06030101
01010C1B030403021 0130906030001
01010A130204040312120A06030101
*
$
```

Figure 22 - Sample vocabulary record.

Using the same notation as introduced earlier, the ordering of data in each frame, shown in figure 22, is: PD(62.5Hz, PD(125Hz), PD(250Hz), PD(500Hz), PD(1000Hz), PD(2000Hz), PD(4000Hz), ZC(Microphone), ZC(4000Hz), ZC(2000Hz), ZC(1000Hz), ZC(500Hz), ZC(250Hz), ZC(125Hz), ZC(62.5Hz). Each record is stored sequentially in the vocabulary file.

After the above initialization, TRAINJFK prompts the user to enter one vocabulary word, or to enter one of three option commands. The first option is to end the training phase, which

ends the TRAINJFK program execution. The last option is to boot the 8751. If the last option is chosen, the program asks the user if a warm boot or a cold boot is desired. Based on the answers given, TRAINJFK will issue the appropriate command to the 8751. The warm boot corresponds to the WBCMD command described earlier. The cold boot corresponds to the BCMD command. The second option allows the user to change the 8751 parameters. If the second option is chosen, the user is shown the current values for SPER, STLTM, the seven thresholds, SPCC and TSAM (see section 4.1). Then, the user is prompted for a new value for each of the above parameters (the values are entered in hexadecimal format). After the parameters are entered, the user is asked two additional questions: are adjacent frames to be averaged? Are five extra frames (after the end of a word has been detected) to be collected? Based on the answers to these questions, the appropriate commands are sent to the 8751. Good values for the parameters are given in Chapter 5.

If a word is to be added to the vocabulary, it is typed in (instead of choosing one of the above options); then TRAINJFK waits for its vocalization. A confirmation prompt is presented when the end of a word is received. This allows the user to reject data created by extraneous noises. When the entire set of words has been entered, the user enters the first option to exit from the program.

4.3 RMATCH5

During the recognition phase, the host computer runs a turbo

pascal program called RMATCH5. As indicated earlier, the recognition phase is split into two parts, the library building part and the recognition part. The library building part is initiated by invoking RMATCH5, assuming the library building files have already been created (using TRAINJFK). When the program is initiated, it first enters an initialization routine where the user is prompted to enter the library disk file name of the library. Once the library words are read into memory from the library disk file, the program asks the user a series of questions.

Question 1: Should the input waves be averaged? If the answer is yes, then the adjacent frames of the words being compared to the library words are averaged together. The averaging is the same as in the LIS program when the ACMD command is issued (previously discussed), but here, the host does the averaging. During the course of the thesis work, it was found that averaging does not improve the recognition rate (see Chapter 5). Therefore, the answer to the question should be no.

Question 2: Should the library wave averaging be done? The answer should be yes, so that when words are matched correctly, the test word (i.e., the word being recognized) is averaged in with the library word that it matched. If the library pattern for a particular word is slightly in error, this averaging is used to correct that pattern. Thus, the wave averaging helps to give the best patterns possible for the library words.

Question 3: Should the results go to the LST device (i.e., the printer)? The answer should be yes, so that all results can be printed out for a permanent record.

Question 4: what mode should the librarian (one of the roles of the RMATCH5 program) be in? Here, the answer should be the automatic mode. In this mode, any test word that is matched wrong (i.e., the test word matches with the wrong library word), or does not match (i.e., the test word does not match with any library word), or is ambiguous (i.e., the test word matched closely with more than one library word) and the closest match is the wrong library word, then the test word is added to the library. In this way, multiple copies of the same word, corresponding to different ways in which the word was uttered, are stored in the library. Also, in this mode, if the test word is ambiguous but the closest match is the right library word, then the test word is averaged with the library word.

After this, RMATCH5 shows its default parameters and then allows the user to adjust them. Good values for these parameters are given in Chapter 5. Finally, the program asks for a list of comparison files (each file created by TRAINJFK). As mentioned before, the files, which are used to build the library (including the initial library file), are known as the library building files. Up to ten files can be specified here (each file terminated by a carriage return). RMATCH5 goes through each file specified, and sequentially uses each word as a test word. During the course of the thesis work, five was found to be a good number of library building files (each file containing one copy of each word). At the conclusion of the RMATCH5 execution, the modified library is stored in the library disk file.

The recognition part of the recognition phase is initiated by invoking RMATCH5, assuming that the test files have already

been made (with TRAINJFK). The library disk file name is the same as before, that is, if it is desired to use the built library. Here, both averaging questions should be answered no, the results should go to the LST device (so that the performance can be recorded on paper), and the librarian should be off (so that the library is not changed). Again, the program parameters should be set as before and finally, up to ten test files (called comparison files by RMATCH5) can be specified.

The matching algorithm (i.e., the program RMATCH5) was developed in ADSL by William Wigger. This matching algorithm does not use the information obtained from the zero crossings in the 62.5Hz, 125Hz, 250Hz, and the 500Hz bands. This does not indicate a lack of information in the zero crossings in the mentioned bands, but simply that there is still room for more experimentation.

Whenever RMATCH5 is invoked, it reads in a library of word patterns from the library disk file. As indicated before, these patterns are known as the library words. RMATCH5 then reads in a test word pattern from a file specified by the user (the comparison file during the library building part, or the test file during the recognition part). The test word is then compared to all the library words using four different matching algorithms. For a more detailed explanation of the following matching algorithms, see the RMATCH5 program listing and see William Wigger's master's thesis [13].

The first matching algorithm, known as match 1, initially calculates the difference in the number of frames between the test word and each library word. Library words which yield

scores above a user specified threshold (FRAME THRESHOLD) are thrown out, or are no longer considered to be possible matches to the test word. In the program, a library word is thrown out by setting its score to a maximum (i.e., 255). Then, the sum of the differences of the normalized average energy in each band, between the test word and each of the library words, is calculated. Library words with scores above a user specified threshold (STAGE 1 THRESHOLD) are thrown out. As indicated before, each library word has a score associated with it. As the matching algorithms are run, they add to each library word's score (i.e., the score is cumulative).

All the library words that are not thrown out by match 1 are then considered by the second matching algorithm, match 2. Here, the time warp scores for all the normalized energy bands, between the test word and the remaining library words, are calculated [1,13]. The term energy bands refers to the outputs of the peak detectors (not the zero crossing detectors). The time warp score is a measure of how similar the patterns are (the lower the score, the more similar the patterns). The time warp scores are added to each library word's score. Library words with scores above a user specified threshold (STAGE 2 THRESHOLD), are thrown out.

All the library words that are not thrown out by match 2 are then considered by the third matching algorithm, match 3. If one library word has a signifigantly lower score than all other library words (STAGE 2 DIFFERENCE) by the end of match 2, then the time warp scores between the test word and this one library word, for the zero crossings in the 1000 Hz, 2000 Hz, and 4000 Hz

bands, are calculated and added. If the sum is lower than a user defined threshold (STAGE 3 THRESHOLD), then the library word is chosen as the matched word (i.e., the word the test word represents). If, on the other hand, more than one library word has a low score by the end of match 2, then the time warp scores between the test word and all the remaining library words, for the zero crossings in the 1000 Hz, 2000 Hz, and 4000 Hz bands, are calculated and added. Here, the sum is added to each library word's score. Library words with scores above a user defined threshold (STAGE 3 THRESHOLD) are thrown out. If at this point one library word has a much lower score than the other library words (STAGE 3 DIFFERENCE), then that library word is chosen as the matched word.

The remaining library words are then considered by the fourth matching algorithm, match 4. If, at the end of match 3, more than one library word has a low score, then the time warp score, between the test word and the two lowest scoring library words, for just the 62.5Hz energy band, is calculated. If one of the library words yields a much lower score (STAGE 4 DIFFERENCE), while the score itself (not the cumulative score), is lower than a user defined threshold (STAGE 4 THRESHOLD), then it is chosen as the matched word. If both library words are still close, then the test word is considered ambiguous. If at some point during the matching, all the library words are thrown out, then the test word gives a no match result.

CHAPTER 5

PERFORMANCE

To evaluate the effectiveness of the voice recognition system, measurements of its performance were made. Here, the results of the measurements are presented, using both live and recorded voices.

Each test used a library that was built using five library building files, each file containing one utterance of each library word (the term library building files was defined in Chapter 3). The values used for the LIS program parameters and the RMATCH5 program parameters are given in Table 1 and Table 2, respectively. (All tables will be shown at the end of this chapter.) The values necessary for running TRAINJFK were given in Chapter 3 (minimum frame count equals five and number of frames to chop off equals ten). The procedures for running the training phase and the recognition phase are also given in Chapter 3.

The testing was done in a quiet room where only the fan noises of an Intel network resource manager were present. The voice used was this author's, except for Tests G and H, where another male speaker made the test. During the testing, the microphone was held about two inches from the mouth or the tape recorder's speaker (depending on the test). However, no attempt was made to precisely maintain this distance. Also, attempts were made by the speakers to pronounce the words as similarly as possible. Two different data sets of words were used during the testing. Data set one consisted of the words, zero, one, two, three, four, five, six, seven, eight, nine, ten, dog. Data set two consisted of the words, north, south, east, west, up, down, right, left, stop, go.

Test A, shown in Table 3, gives the results of recognizing data set one the same day that the library was built (i.e., the test files were made the same day the library was built). Here, the library was also made up of words from data set one. The column labeled RIGHT shows how many times the corresponding test word was recognized correctly. The column labeled WRONG shows how many times the corresponding test word was recognized incorrectly (i.e., the test word matched to the wrong library word). The columns labeled AMBIGUOUS and NO MATCH show how many times the corresponding test word yielded an ambiguous result or a no match result, respectively; (both of these were defined in Chapter 3). Test B, shown in Table 4, gives the results of recognizing data set one, one day after the library was built (i.e., the test files were taken the day after the library building files were taken). Here, the same library was used as

in Test A.

Test C, shown in Table 5, is the same as Test A, except that Test C uses data set two instead of data set one. Again, the recognition occurred the same day that the library was built. Also, the library used was built from words in data set two. Test D, shown in Table 6, gives the results of recognizing data set two, one day after the library was built. This test uses the same library as was used in Test C.

Test E, shown in Table 7, gives the results of recognizing recorded utterances. That is, a recording was made of data set two, and the recording was used to create 5 files with which to build the library. The same recording was used to create the test files. The results of the test show that the percentage of correct recognition is not 100 percent. This is probably due to faulty word boundary detection during the recognition phase (e.g., the tape recorder made some noise when it was turned on or off). Test F, shown in Table 8, is the same as Test E, except that data set one was used instead of data set two.

Test G, shown in Table 9, is exactly like Test A, except that Test G uses a different speaker (other than the author). As in Test A, the recognition occurred the same day that the library was built. Again, the library used was built from the same data set as the test files were made from (data set one). Test H, shown in Table 10, is the same as Test G, except that the recognition occurred two days after the library was built.

Test I, shown in Table 11, is the same as Test C, except that LIS had its averaging function on during Test I. Test J, shown in Table 12, is exactly like Test D, except that LIS had

its averaging function on. Comparing the results of Test I with the results of Test C, and also comparing the results of Test J with the results of Test D, it seems that the averaging function improves the percentage of correct recognition. However, if the same data are processed both with and without the averaging (i.e., the averaging is performed by RMATCH5), no improvement in the percentage of recognition is achieved. Thus, the data taken for Tests I and J are more consistent than the data taken for Tests C and D (i.e., the speaker pronounced the words more similarly for Tests I and J). Therefore, it is concluded that averaging adjacent frames does not increase the percentage of recognition. Even though the favorable results shown in Tests I and J could not prove that averaging helps the percentage of correct recognition, they do show the capability of the recognition system for data set two.

The results presented indicate that the performance of the system depends on several factors: on the words chosen for the library and on how consistently the speaker pronounces the words. In most of the cases presented here, the percentage of recognition is lower when the recognition takes place sometime after the building of the library. Presumably, this is because the speaker pronounces the words differently from how the words were pronounced during the building of the library.

TABLE 1 - LIS PROGRAM PARAMETERS

SPER	= E800	THR5	= 0A
STLTM	= 01	THR6	= 0A
THR1	= 0A	THR7	= 03
THR2	= 0A	SPCC	= 0A
THR3	= 0A	TSAM	= 03
	THR4 = 0A		

AVERAGE ADJACENT FRAMES = N (except for tests I and J)
COLLECT FIVE EXTRA FRAMES = N

TABLE 2 - RMATCH5 PROGRAM PARAMETERS

(1) FRAME THRESHOLD	= 10
(2) STAGE 1 THRESHOLD	= 40
(3) STAGE 2 THRESHOLD	= 25
(4) STAGE 2 DIFFERENCE	= 8
(5) STAGE 3 THRESHOLD	= 10
(6) STAGE 3 DIFFERENCE	= 5
(7) STAGE 4 THRESHOLD	= 10
(8) STAGE 4 DIFFERENCE	= 4
(9) AVERAGING WEIGHT FACTOR	= 4

TABLE 3 - TEST A, LIBRARY BUILT THE SAME DAY

TEST WORD	NUMBER OF TRIES	RIGHT	WRONG	AMBIGUOUS	NO MATCH
Zero	5	5	0	0	0
One	5	4	0	0	1
Two	5	4	1	0	0
Three	5	4	1	0	0
Four	5	2	1	2	0
Five	5	5	0	0	0
Six	5	5	0	0	0
Seven	5	4	1	0	0
Eight	5	5	0	0	0
Nine	5	5	0	0	0
Ten	5	5	0	0	0
Dog	5	5	0	0	0

RIGHT	= 53/60	(88%)
WRONG	= 4/60	(7%)
AMBIGUOUS	= 2/60	(3%)
NO MATCH	= 1/60	(2%)

TABLE 4 - TEST B, PREVIOUSLY BUILT LIBRARY

TEST WORD	NUMBER OF TRIES	RIGHT	WRONG	AMBIGUOUS	NO MATCH
Zero	10	10	0	0	0
One	10	8	0	2	0
Two	10	9	0	1	0
Three	10	10	0	0	0
Four	10	6	1	3	0
Five	10	9	0	1	0
Six	10	10	0	0	0
Seven	10	5	3	1	1
Eight	10	10	0	0	0
Nine	10	6	1	3	0
Ten	10	4	0	6	0
Dog	10	8	0	2	0

RIGHT = 95/120 (79%)
WRONG = 5/120 (4%)
AMBIGUOUS = 19/120 (16%)
NO MATCH = 1/120 (1%)

TABLE 5 - TEST C, LIBRARY BUILT THE SAME DAY

TEST WORD	NUMBER OF TRIES	RIGHT	WRONG	AMBIGUOUS	NO MATCH
North	5	3	0	2	0
South	5	0	2	3	0
East	5	3	0	0	2
West	5	3	2	0	0
Up	5	4	1	0	0
Down	5	3	1	1	0
Right	5	4	0	1	0
Left	5	3	0	2	0
Stop	5	4	0	0	1
Go	5	3	0	1	1

RIGHT = 30/50 (60%)
WRONG = 6/50 (12%)
AMBIGUOUS = 10/50 (20%)
NO MATCH = 4/50 (8%)

TABLE 6 - TEST D, PREVIOUSLY BUILT LIBRARY

TEST WORD	NUMBER OF TRIES	RIGHT	WRONG	AMBIGUOUS	NO MATCH
North	10	7	3	0	0
South	10	5	2	2	1
East	10	10	0	0	0
West	10	8	0	2	0
Up	10	9	1	0	0
Down	10	3	0	7	0
Right	10	10	0	0	0
Left	10	6	2	2	0
Stop	10	10	0	0	0
Go	10	10	0	0	0

RIGHT = 78/100 (78%)
WRONG = 8/100 (8%)
AMBIGUOUS = 13/100 (13%)
NO MATCH = 1/100 (1%)

TABLE 7 - TEST E, RECORDED VOICE

TEST WORD	NUMBER OF TRIES	RIGHT	WRONG	AMBIGUOUS	NO MATCH
North	5	5	0	0	0
South	5	5	0	0	0
East	5	5	0	0	0
West	5	4	0	0	1
Up	5	5	0	0	0
Down	5	5	0	0	0
Right	5	4	0	0	1
Left	5	5	0	0	0
Stop	5	4	1	0	0
Go	5	5	0	0	0

RIGHT = 47/50 (94%)
WRONG = 4/50 (2%)
AMBIGUOUS = 0/50 (0%)
NO MATCH = 2/50 (4%)

TABLE 8 - TEST F, RECORDED VOICE

TEST WORD	NUMBER OF TRIES	RIGHT	WRONG	AMBIGUOUS	NO MATCH
Zero	5	5	0	0	0
One	5	5	0	0	0
Two	5	5	0	0	0
Three	5	5	0	0	0
Four	5	4	0	1	0
Five	5	5	0	0	0
Six	5	4	0	0	1
Seven	5	5	0	0	0
Eight	5	5	0	0	0
Nine	5	5	0	0	0
Ten	5	5	0	0	0
Dog	5	5	0	0	0

RIGHT = 58/60 (96%)
WRONG = 0/60 (0%)
AMBIGUOUS = 1/60 (2%)
NO MATCH = 1/60 (2%)

TABLE 9 - TEST G, LIBRARY BUILT THE SAME DAY

TEST WORD	NUMBER OF TRIES	RIGHT	WRONG	AMBIGUOUS	NO MATCH
Zero	6	3	0	0	3
One	6	4	0	0	2
Two	6	2	1	0	3
Three	6	6	0	0	0
Four	6	5	1	0	0
Five	6	5	0	1	0
Six	6	4	0	0	2
Seven	6	2	2	0	2
Eight	6	3	0	0	3
Nine	6	6	0	0	0
Ten	6	6	0	0	0
Dog	6	6	0	0	0

RIGHT = 52/72 (72%)
WRONG = 4/72 (6%)
AMBIGUOUS = 1/72 (1%)
NO MATCH = 15/72 (21%)

TABLE 10 - TEST H, PREVIOUSLY BUILT LIBRARY

TEST WORD	NUMBER OF TRIES	RIGHT	WRONG	AMBIGUOUS	NO MATCH
Zero	10	5	3	0	2
One	10	8	0	1	1
Two	10	1	3	0	6
Three	10	9	0	1	0
Four	10	5	2	0	3
Five	10	10	0	0	0
Six	10	5	1	0	4
Seven	10	4	5	1	0
Eight	10	6	0	0	4
Nine	10	10	0	0	0
Ten	10	10	0	0	0
Dog	10	9	0	1	0

RIGHT = 82/120 (68%)
WRONG = 14/120 (12%)
AMBIGUOUS = 4/120 (3%)
NO MATCH = 20/120 (17%)

TABLE 11 - TEST I, LIBRARY BUILT THE SAME DAY

TEST WORD	NUMBER OF TRIES	RIGHT	WRONG	AMBIGUOUS	NO MATCH
North	5	3	0	2	0
South	5	5	0	0	0
East	5	5	0	0	0
West	5	5	0	0	0
Up	5	5	0	0	0
Down	5	5	0	0	0
Right	5	5	0	0	0
Left	5	4	0	1	0
Stop	5	4	1	0	0
Go	5	5	0	0	0

RIGHT = 46/50 (92%)
WRONG = 1/50 (2%)
AMBIGUOUS = 3/50 (6%)
NO MATCH = 0/50 (0%)

TABLE 12 - TEST J, PREVIOUSLY BUILT LIBRARY

TEST WORD	NUMBER OF TRIES	RIGHT	WRONG	AMBIGUOUS	NO MATCH
North	10	8	0	2	0
South	10	7	2	0	1
East	10	9	0	0	1
West	10	7	0	3	0
Up	10	10	0	0	0
Down	10	10	0	0	0
Right	10	10	0	0	0
Left	10	7	1	2	0
Stop	10	9	0	0	1
Go	10	10	0	0	0

RIGHT = 87/100 (87%)
WRONG = 3/100 (3%)
AMBIGUOUS = 7/100 (7%)
NO MATCH = 3/100 (3%)

CHAPTER 6

CONCLUDING REMARKS

The goal of this thesis research, described in Chapter 1, has been met, that is, the ADSL voice recognition system has been improved so as to allow for more sophisticated software experiments. As described in Chapter 3, the design of the feature extractor has emphasis on hardware and on inexpensive off-the-shelf components, as the goal states. The use of a digital signal processor was considered. However, one was not available at the time. Also, the option of accomplishing the feature extraction (filtering and peak detection) within the host computer was discarded because of the increased processing time. This was exacerbated by the extreme unfriendliness of the initial host computer.

Currently, much of the research in voice recognition uses an approach that involves Linear Predictive Coding (LPC coding). Some success has been achieved using this method: however, it

gives trouble when noise is present with the speech signal. Even though noise was not considered here, other research dealing with the analysis of speech in the presence of noise uses the same general approach (i.e., similar feature extraction) as in this thesis [14]. Therefore, the thesis can be considered as a contribution to a continuing series of current research projects on voice recognition.

Looking at the results of the tests done on the system, shown in Chapter 5, it is evident that the system does have some weak points. In particular, the word boundary detection algorithm could be revised to include one set of thresholds for the beginning of word detection and another set of thresholds for the end of word detection. Another possibility would be to use the zero crossing information to help detect the word boundaries.

The feature extractor, as it stands, can be used for further experimentations for word boundary detection and noise analysis. The experimentation is made possible by the commands that the feature extractor can recognize (e.g., a command that tells the feature extractor to skip the word boundary detection algorithm and to start taking in data). The commands are explained fully in the LIS program listing.

It is the author's wish that this work will stimulate future students to make their own contributions to ADSL's voice recognition system.

APPENDIX A

SCHEMATIC DIAGRAMS

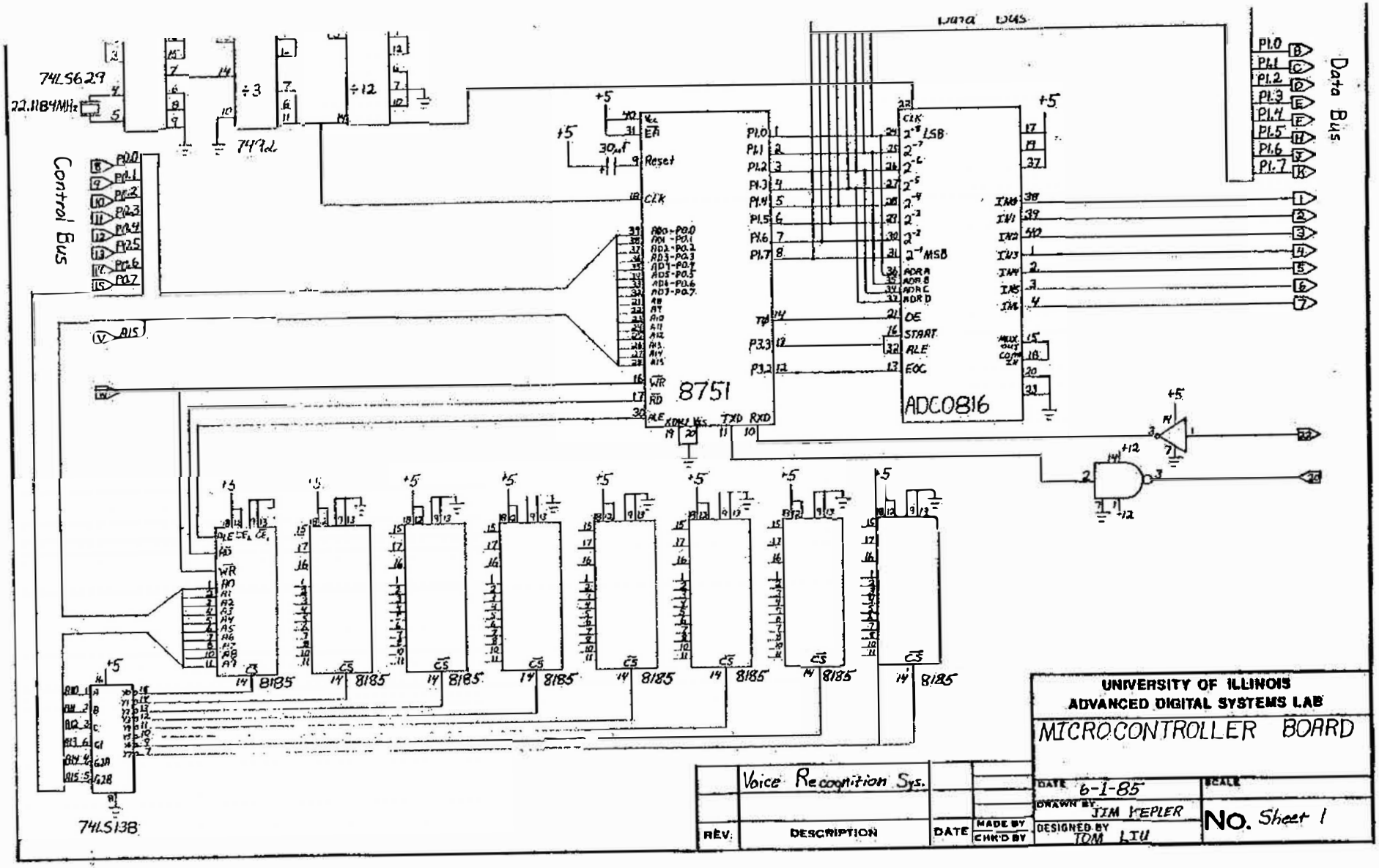


Figure A1 - Microcontroller board.

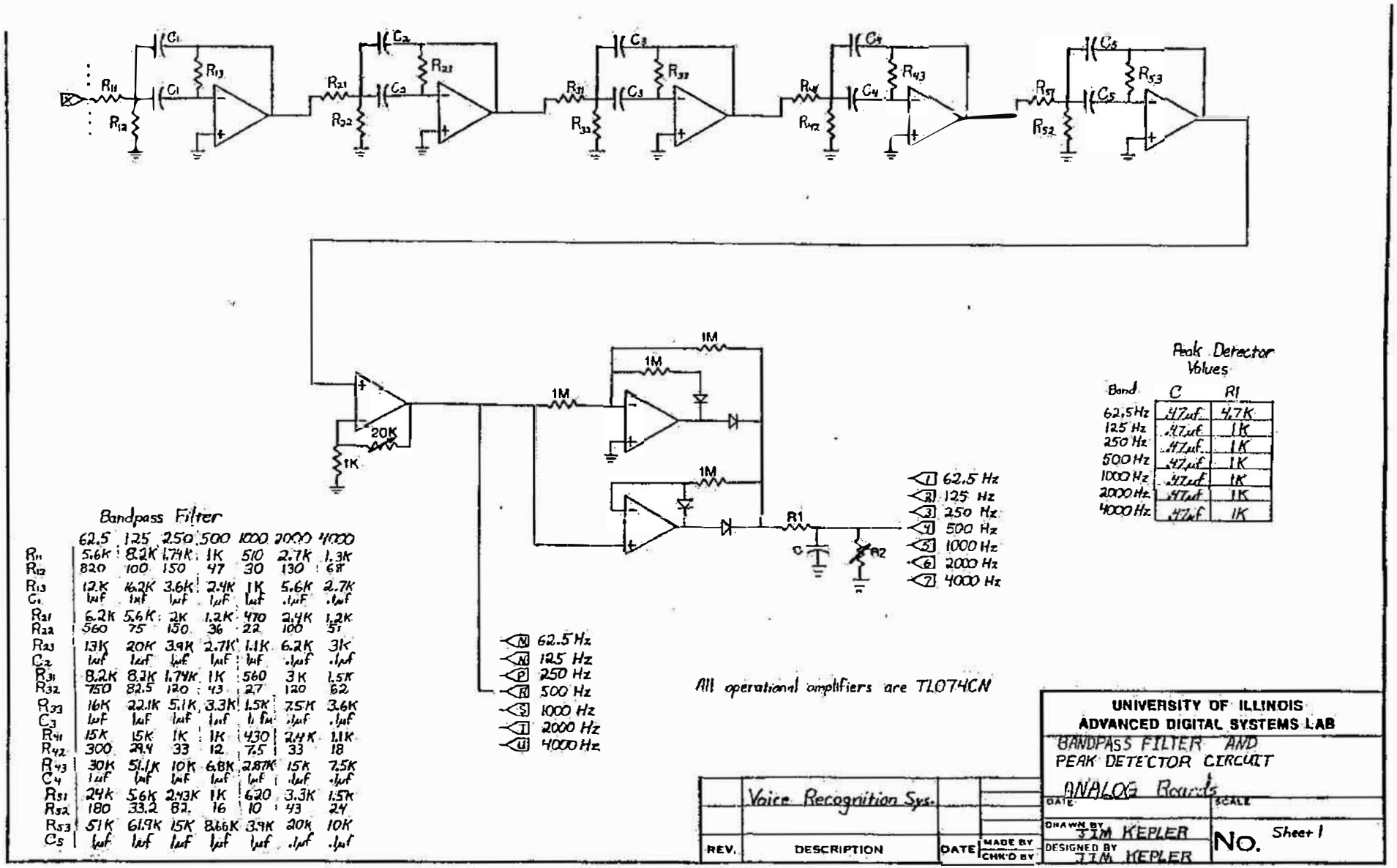
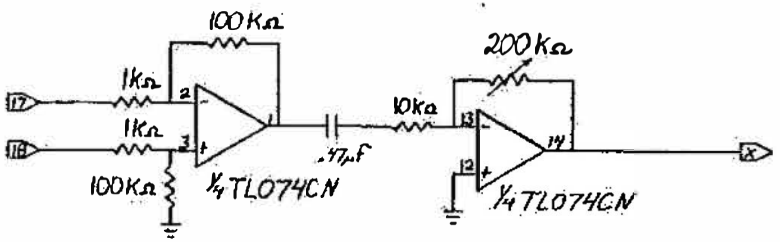


Figure A2 - Analog boards.



	Voice Recognition Sys			UNIVERSITY OF ILLINOIS ADVANCED DIGITAL SYSTEMS LAB	
				MICROPHONE AMPLIFIER	
REV.	DESCRIPTION	DATE	MADE BY / CHK'D BY	DATE 6-1-85 / DRAWN BY JIM KEPLER / DESIGNED BY TOM LIU	SCALE No. Sheet

Figure A3 - Microphone amplifier.

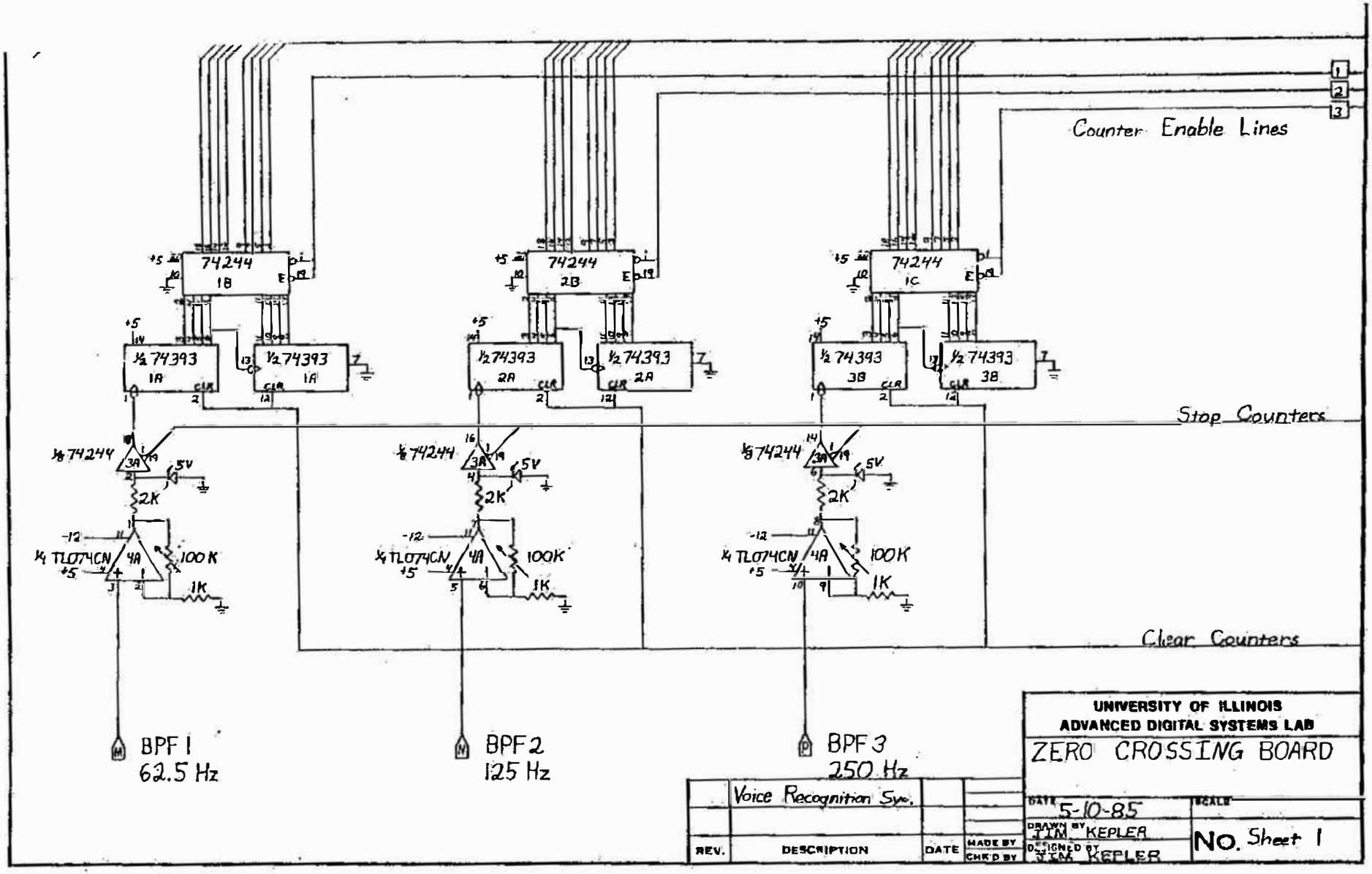


Figure A4 - Zero crossing board, sheet 1.

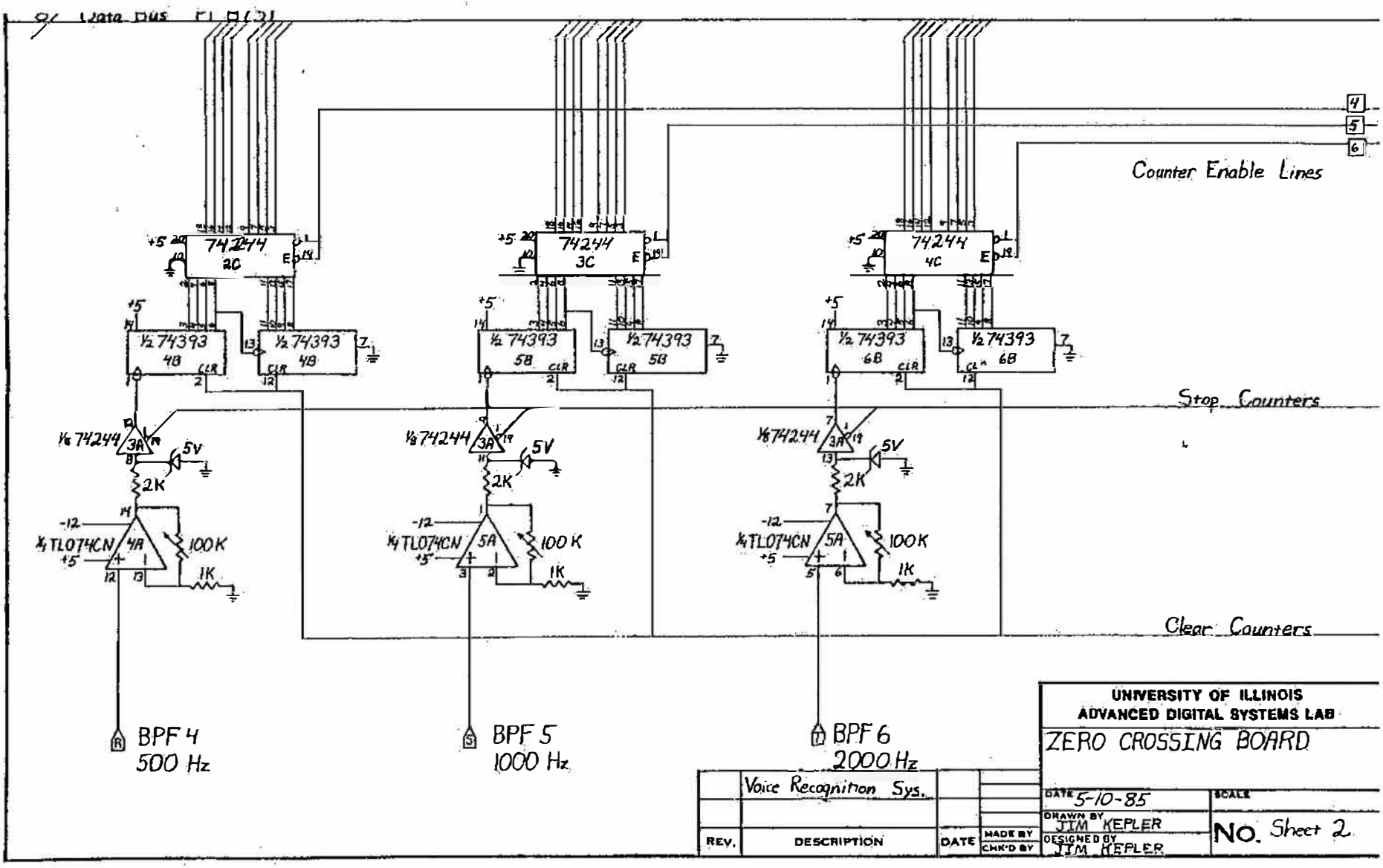


Figure A5 - Zero crossing board, sheet 2.

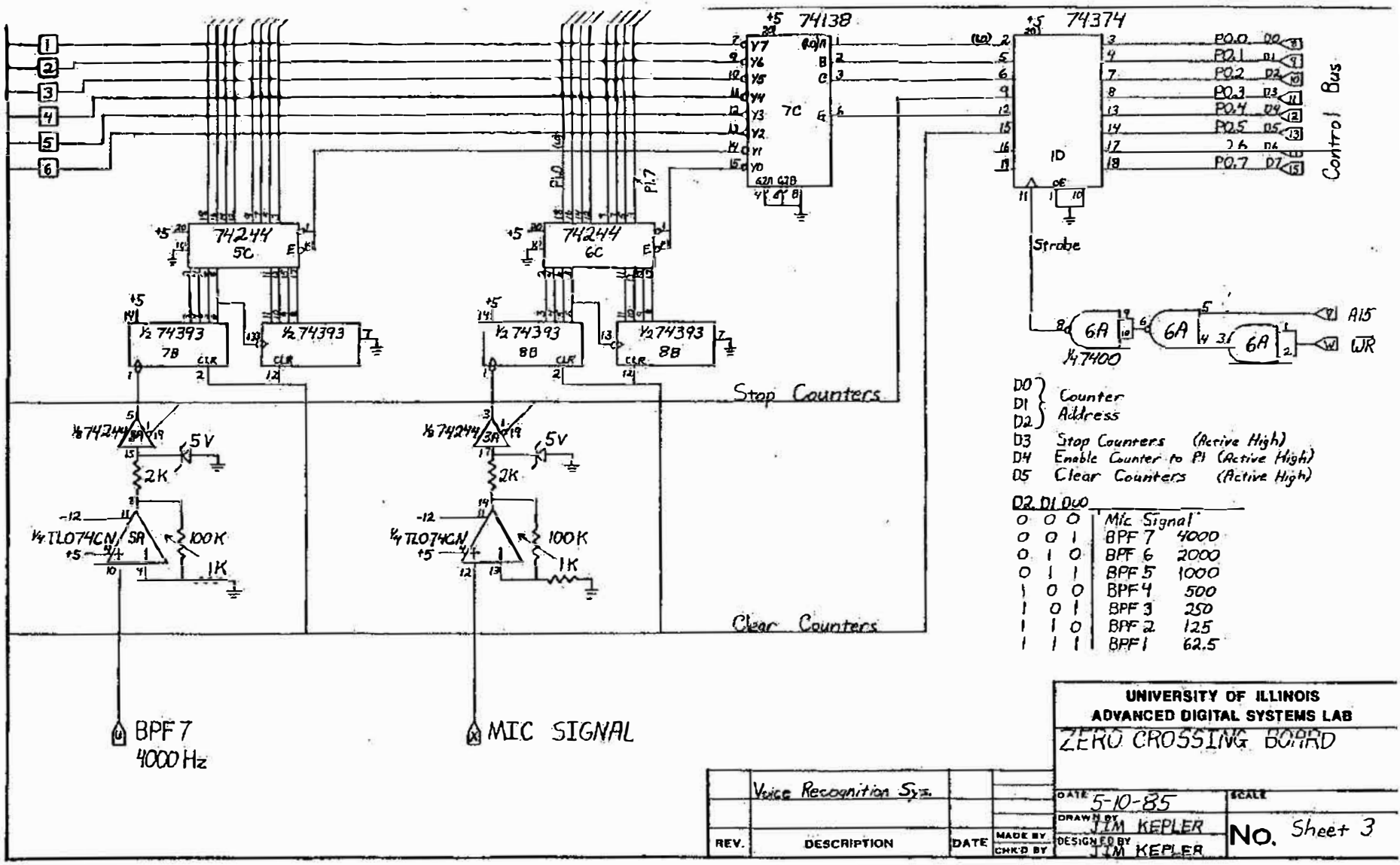


Figure A6 - Zero crossing board, sheet 3.

APPENDIX B

DESIGN OF BANDPASS FILTERS

As mentioned previously, the design of the Butterworth bandpass filters is relatively straightforward [8]. The following will outline the procedure used to design the bandpass filters.

Step 1

Determine the specifications for the bandpass filter in terms of attenuation, as shown in figure B1. The frequency should be specified in units of radians per second. In this design, a(min)=35 dB and a(max)=.1 dB. In determining the

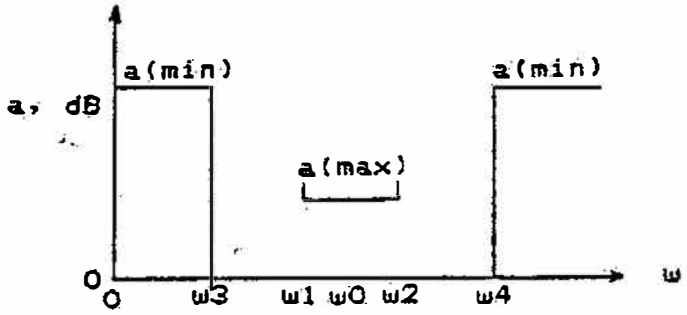


Figure B1 - Filter specifications.

frequencies w1, w2, w3, w4, and w0, equation (1) must be

$$\omega_1\omega_2 = \omega_3\omega_4 = \omega_0^2 \qquad (1)$$

satisfied. In this design, w3 and w4 are specified, leaving w1 and w2 to be determined by iteration. That is, a guess is made for w1, then equation (1) is used to determine w2. Step 3 gives the filter order resulting from the guess of w1. If the filter order is not ten, then another guess for w1 is made. The values used for w3 and w4 and the resulting values for w1 and w2 are shown in figure B2.

BAND	w3 (Hz)	w4 (Hz)	w1 (Hz)	w2 (Hz)
62.5Hz	30	100	46	65
125Hz	100	150	116	129
250Hz	150	350	200	262
500Hz	350	650	434	525
1000Hz	650	1350	836	1049
2000Hz	1350	2650	1705	2100
4000Hz	2650	5350	3378	4201

Figure B2 - Frequency specifications.

Step 2

Specify the lowpass filter prototype, using equations (2) and (3) and figure B3. Note that a(max) and a(min) are the same as in the bandpass specifications.

$$W_P = 1 \tag{2}$$

$$W_S = \frac{w4 - w3}{w2 - w1} \tag{3}$$

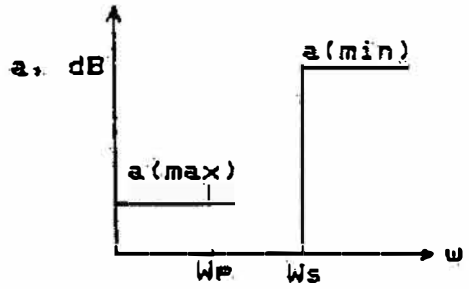


Figure B3 - Lowpass prototype.

Step 3

Evaluate equation (4) and round up to the nearest integer. The integer gives the order of the lowpass prototype, which is one-half the order of the bandpass filter. Evaluate equation

$$n = \frac{\log\left[\left(10^{a(min)/10} - 1\right) \Big/ \left(10^{a(max)/10} - 1\right)\right]}{2 \log [Ws/Wp]} \qquad (4)$$

$$WO = \left[10^{a(max)/10} - 1\right]^{-(1/2)n} \qquad (5)$$

(5), using the integer found in equation (4), in preparation for Step 4. For this design, n is always five and WO is 1.45639 for each filter. WO is constant because Ws turns out to be the same for the filters being designed.

Step 4

Find the poles of the Butterworth lowpass prototype. This is done by finding the Butterworth angles, B, and then realizing that all the poles lie on a circle of radius WO (from equation (5)) centered at the origin in the s-plane. The Butterworth angles are determined using two rules:

1. If n is odd, then there is a pole at B=0 degrees; if n is even, then there are poles at B= +/- 90/n degrees.
2. Poles are separated by 180/n degrees, and all the poles are in the left half plane.

The Butterworth angles are measured with respect to the negative real axis, as shown in figure B4. For this design, the five lowpass prototype pole locations for each filter are shown in figure B5.

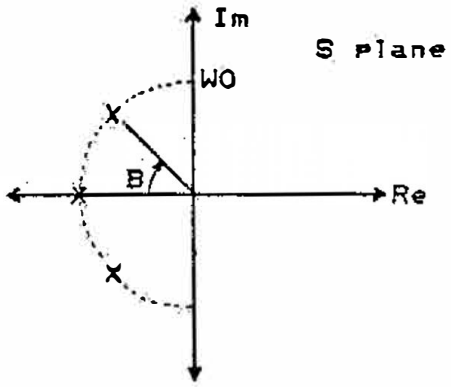


Figure B4 - Butterworth angle for n = 3.

-1.45639
-1.17824 +/- J .85604
-.45005 +/- J 1.38511

Figure B5 - Lowpass prototype poles.

Step 5

The bandpass filter pole locations are determined using the Geffe algorithm. The algorithm transforms each lowpass prototype pole pair into four bandpass poles and each real lowpass pole into a bandpass pole pair. Therefore, each lowpass prototype pole becomes two bandpass poles. If the pole to be transformed is a real pole located at -S1, then the bandpass pole pair is located at Butterworth angles given by equation (7), and on a circle of radius w0 (as determined by equation (1)). If the pole to be transformed is a pole pair and thus complex, located at S2 +/- J R2, then equations (8) through (17) are evaluated. The resulting two bandpass pole pairs are located on two circles, of radius w01 and w02 (given by equations (15) and (16)), at Butterworth angles given by equation (17). However, for the circuit realization, the values needed are Q, w01 and w02. The

$$q = \frac{w0}{w2 - w1} \tag{6}$$

$$B = \cos^{-1}(S1/2q) \tag{7}$$

$$C = S2^2 + R2^2 \tag{8}$$

$$D = 2\,(S2)/q \tag{9}$$

$$E = 4 + C/q^2 \tag{10}$$

$$G = \sqrt{E^2 - 4D^2} \tag{11}$$

$$Q = (1/D)\sqrt{1/2\,(E + G)} \tag{12}$$

$$K = (S2)\,Q/q \tag{13}$$

$$W = K + \sqrt{K^2 - 1} \tag{14}$$

$$w02 = W\,(w0) \tag{15}$$

$$w01 = (1/W)\,w0 \tag{16}$$

$$B = \cos^{-1}\left[1/(2Q)\right] \tag{17}$$

w01 and w02 values are the w values shown in figure B5. The Q and w values calculated for this thesis are shown in figure B5. Each stage represents a pole pair, the Q value gives the Butterworth angle, and the w value gives the radius of the circle that the poles are on.

FILTER	STAGE 1	STAGE 2	STAGE 3	STAGE 4	STAGE 5
62.5Hz	Q=2.01646 w=343.6	Q=2.51990 w=398.5	Q=2.51990 w=269.3	Q=6.70537 w=434.3	Q=6.70537 w=271.8
125Hz	Q=6.60108 w=769.1	Q=8.16753 w=804.2	Q=8.16753 w=735.6	Q=21.4169 w=826.5	Q=21.4169 w=715.7
250Hz	Q=2.56817 w=1440	Q=3.19568 w=1616.5	Q=3.19568 w=1282.8	Q=8.45252 w=1731.7	Q=8.45252 w=1197.5
500Hz	Q=3.59816 w=2997	Q=4.46258 w=3253.4	Q=4.46258 w=2760.8	Q=11.7453 w=3419.5	Q=11.7453 w=2626.7
1000Hz	Q=3.0228 w=5886	Q=3.7542 w=6491.7	Q=3.7542 w=5337	Q=9.9015 w=6886	Q=9.9015 w=5031
2000Hz	Q=3.2956 w=11884	Q=4.0898 w=13000	Q=4.0898 w=10864	Q=10.7743 w=13724	Q=10.7743 w=10291
4000Hz	Q=3.1579 w=23658	Q=3.9203 w=25982	Q=3.9203 w=21542	Q=10.3334 w=27492	Q=10.3334 w=20359

Figure B5 - Filter specifications.

Step 6

After the Q and w values are determined for each stage of each filter, the circuit realization can be completed. Each stage is realized using a Friend circuit, as shown in figure B6.

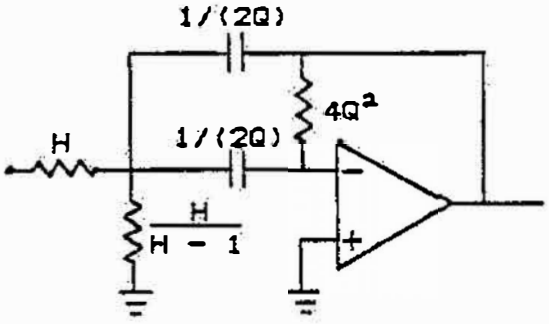


Figure B6 - Friend circuit.

The value of H is determined by equation (18), where w0 is the center frequency of the bandpass (given by equation (1)), and w

is the frequency associated with the particular stage (e.g., the w values given in figure B4). This will ensure that the gain of each stage will be one at the center frequency of the bandpass filter. Next, the component values are scaled using equations (19) and (20). The old value of the capacitor is Cold while Cnew

$$1/H = \sqrt{\frac{(2\ Q\ w\ w0)^2}{(w^2 - w0^2)^2 + (w\ w0/Q)^2}} \qquad (18)$$

$$Cnew = Cold/(Kf\ Km) \qquad (19)$$

$$Rnew = Km\ Rold \qquad (20)$$

is the new scaled value of the capacitor. Also, Rnew is the scaled resistance value while Rold is the unscaled value. Kf is the frequency scaling and must be set equal to the frequency associated with the stage, w (i.e., there is a different value of Kf for each stage of each filter). Km is the magnitude scaling and is chosen to give convenient component values. The final component values for all stages of all the filters are given in the schematic diagram in Appendix A.

APPENDIX C

BACKPLANE PIN ASSIGNMENTS

TABLE C1 - BACKPLANE PIN ASSIGNMENTS

1	MUX Ch 0, PD 62.5Hz	A	+5 V
2	MUX Ch 1, PD 125Hz	B	P1.0 (data bus)
3	MUX Ch 2, PD 250Hz	C	P1.1 (data bus)
4	MUX Ch 3, PD 500Hz	D	P1.2 (data bus)
5	MUX Ch 4, PD 1000Hz	E	P1.3 (data bus)
6	MUX Ch 5, PD 2000Hz	F	P1.4 (data bus)
7	MUX Ch 6, PD 4000Hz	H	P1.5 (data bus)
8	P0.0 (control bus)	J	P1.6 (data bus)
9	P0.1 (control bus)	K	P1.7 (data bus)
10	P0.2 (control bus)	L	P3.5
11	P0.3 (control bus)	M	BPF 1 62.5Hz
12	P0.4 (control bus)	N	BPF 2 125Hz
13	P0.5 (control bus)	P	BPF 3 250Hz
14	P0.6 (control bus)	R	BPF 4 500Hz
15	P0.7 (control bus)	S	BPF 5 1000Hz
16		T	BPF 6 2000Hz
17	MIC 1	U	BPF 7 4000Hz
18	MIC 2	V	A15
19	-12 V	W	WR/
20	Serial Out	X	MIC (amplified)
21	+12 V	Y	
22	Serial In	Z	GND

APPENDIX D

8751 LIS PROGRAM

```
ISIS-II MCS-51 MACRO ASSEMBLER V2.0
OBJECT MODULE PLACED IN :F1:LIS.OBJ
ASSEMBLER INVOKED BY:  asm51 :f1:lis

LOC  OBJ            LINE     SOURCE

                       1     ;------------------------------------------------------------------------
                       2     ;
                       3     ;
                       4     ;*************************************************************************
                       5     ;**********************                        ************************
                       6     ;********************** voice recognition  ************************
                       7     ;**********************       system       ************************
                       8     ;**********************                    ************************
                       9     ;**********************        LIS         ************************
                      10     ;**********************                    ************************
                      11     ;*************************************************************************
                      12     ;
                      13     ;
                      14     ;------------------------------------------------------------------------
                      15     ;
                      16     ;
                      17     ;
                      18     ;
                      19     ;       Date:           May 5, 1983
                      20     ;       Update:         July 6, 1983 - Revision 3.21
                      21     ;       Update:         February 19, 1984 - Revision 4.10
                      22     ;       Update:         October 8, 1984 - Revision 4.21
                      23
                      24     ;
                      25     ;       Author:         Jim Kepler
                      26     ;
                      27     ;       Source:         LIS.SRC
                      28     ;       Object:         LIS.OBJ
                      29     ;
                      30     ;       Purpose:
                      31     ;                       This program resides in the 8751, which is the
                      32     ;                       microcontroller in the feature extracter (which
                      33     ;                       is part of the voice recognition system).
                      34     ;                       The program is responsible for scanning the
                      35     ;                       outputs of the peak detectors (PD) and the zero
                      36     ;                       crossing detectors (ZC), detecting the
                      37     ;                       boundaries of words, and communicating with
                      38     ;                       the host computer.
                      39     ;
                      40     ;       Note:
                      41     ;                       This version of the program is based on and
                      42     ;                       modified from earlier programs:
                      43     ;                       TEST5.SRC       - Jim Kepler (EE 246 - Spring 1983)
                      44     ;                       VOICE.SRC       - Carl Steiner (EE 246 - Spring 1983)
                      45     ;                       LISTEN.SRC      - Thomas Liu  (Masters thesis Su 83)
```

```
LOC  OBJ            LINE     SOURCE

                      46     ;                       LISTEN.SRC      - Trang Nguyen (EE 246 - Spring 1984)
                      47     ;                       LIS             - Jim Kepler (Masters thesis Spring 1985)
                      48     ;
                      49     ;
                      50     ;

                      51     ;-----------------------------------------------------------------------
                      52     ;
                      53     ;
                      54     ;                               *** BLOCK COMMENT ***
                      55     ;
                      56     ;                       This is an 8751 program that utilizes the serial
                      57     ;                       port.  The clock frequency of the 8751 should be
                      58     ;                       7.3728 MHz, in order that the baud rate comes out
                      59     ;                       correct (9600 baud).  The power of this program
                      60     ;                       is awesome, as it was designed for experimental
                      61     ;                       purposes.  However, when the program is executed
                      62     ;                       in conjunction with the voice recognition system
                      63     ;                       only a fraction of its power is utilized, as most
                      64     ;                       of the "power" was used for experimental purposes.
                      65     ;                       Throughout the comments, the term  frame is used
                      66     ;                       frequently.  A frame is the collection of data from
                      67     ;                       each of the peak detectors and each of the zero
                      68     ;                       crossing detectors, at one particular time.  In the
                      69     ;                       external buffer each frame consists of 31 bytes,
                      70     ;                       30 ascii hexadecimal data bytes and a NL character,
                      71     ;                       which signifies the end of a frame.  The external
                      72     ;                       buffer is the external data memory.  The temporary
                      73     ;                       buffers (there are two, one at TBNEW and one at TBOLD)
                      74     ;                       are in the internal data ram (in the 8751).  In the
                      75     ;                       comments the following abbreviations are used:
                      76     ;
                      77     ;                               ZC=     zero crossing counter
                      78     ;                               BPF=    bandpass filter
                      79     ;                               PD=     peak detector
                      80     ;
                      81     ;                       Note, a frame can consist of just an end of word
                      82     ;                       marker or an end of file marker.
                      83     ;
                      84     ;-----------------------------------------------------------------------;
                             ******* CONSTANTS *******
                      85     ;
                      86     ;
                      87     ;
                      88     ; BPF   = Number of bandpass filters.
                      89     ;
                      90     ; STCMD = Command from host to start looking for a word.
                      91     ;
                      92     ; RDCMD = Command from host to transmit one frame of data.
                      93     ;
```

```
LOC  OBJ            LINE     SOURCE

                      94     ; PCMD  = Command from host to go into parameter changing routine
                      95     ;         PARAM.
                      96     ;
                      97     ; DSPCMD= Command from host to stop taking in frames of data,
                      98     ;         and put an end of file marker into the external
                      99     ;         buffer.  Data will still be output through the serial
                     100     ;         port upon request (this is a data stop command).
                     101     ;
                     102     ; DSCMD = Command from host to start taking in frames of data
                     103     ;         after the data input was stopped by the DSPCMD command.
                     104     ;         (this is a data start command).
                     105     ;
                     106     ; GDCMD = Command from host to start gathering frames of data
                     107     ;         at the system sample rate (default is 10ms) until the
                     108     ;         external buffer  overflows or the data input is stopped
                     109     ;         by the DSPCMD or SDCMD command.  Note this gathers
                     110     ;         data unconditionally (no boundary detection is used).
                     111     ;
                     112     ; SDCMD = Command from host to stop the gathering of frames of data
                     113     ;         initiated by the GDCMD.  This causes an end of word condition.
                     114     ;
                     115     ; WBCMD = Command from host to warm boot the system.  The external
                     116     ;         buffer pointers are reset, the temporary buffer pointers are
                     117     ;         reset, all the flags are reset and the program starts from
                     118     ;         the beginning.
                     119     ;
                     120     ; BCMD  = Command from host to boot the system.  This does everything
                     121     ;         the warm boot does and in addition sets all the program
                     122     ;         parameters back to their default values.
                     123     ;
                     124     ; GTSCMD= Command from host to give it one frame of data.
                     125     ;         This does not look for a beginning of a word, in fact the
                     126     ;         DETECT routine is skipped when this command is invoked.
                     127     ;
                     128     ; ACMD  = Command from host to tell 8751 to average every two adjacent
                     129     ;         frames before they are converted and stored into the external
                     130     ;         data buffer.  This does not reduce the data rate, it is only
                     131     ;         a smoothing function.  Note, since the first frame collected
                     132     ;         is averaged against a zero frame (there is no previous frame),
                     133     ;         actually one-half the first frame is stored in the external
                     134     ;         buffer.  However, since the first frame is rarely included as
                     135     ;         part of a word, this is no problem.
                     136     ;
                     137     ; DACMD = Command from host to tell the 8751 not to average adjacent
                     138     ;         frames.  This command nullifies the ACMD command.
                     139     ;
                     140     ; TSCMD = Command from host to tell the 8751 to collect #NTSS (default
                     141     ;         is 5) frames of data after the end of the word has been detected.
                     142     ;         These data are placed after the end of word marker but before the
                     143     ;         end of file marker in the external buffer.
```

```
LOC  OBJ          LINE    SOURCE

                   144    ;
                   145    ; NTSCMD= Command from host to tell the 8751 not to collect any frames
                   146    ;         of data after the end of a word has been declared.  Note, this
                   147    ;         command nullifies the TSCMD command.
                   148    ;
                   149    ; EOF   = End of file marker.
                   150    ;
                   151    ; EOWM  = End of word marker.
                   152    ;
                   153    ; NL    = End of frame marker.
                   154    ;
                   155    ; XMBEG = Beginning of External Buffer Memory.
                   156    ;
                   157    ; XMEND = End of External Buffer Memory + 1.
                   158    ;
                   159    ; MUXST = Address of the first channel of the multiplexer.
                   160    ;
                   161    ;
                   162    ;
                   163    ;-----------------------------------------------------------------------
                   164    ;
                   165    ;
                   166    ;
                   167    ;                ******* EQUATES STATEMENT *******
                   168    ;
                   169    ;
0000               170    CR      EQU     0DH             ; Carriage return
                   171    ;
0020               172    SPACE   EQU     20H             ; Space
                   173    ;
0007               174    BPF     EQU     7               ; Number of bandpass filters
                   175    ;
000F               176    BPFZC   EQU     0FH             ; Number of BPF plus ZC counters
                   177    ;
                   178    ; Commands from the host computer.
                   179    ;
0021               180    STCMD   EQU     '!'             ; Start searching for a word
                   181    ;
000A               182    RQCMD   EQU     10              ; Request for next frame of data
                   183    ;                                 this is a NL.
                   184    ;
0023               185    PCMD    EQU     '#'             ; Go to PARAM (parameter) routine
                   186    ;
0028               187    DSPCMD  EQU     '('             ; Suspend data input
                   188    ;
0029               189    DSCMD   EQU     ')'             ; Resume data input after DSPCMD
                   190    ;
0022               191    GDCMD   EQU     '"'             ; Start gathering frames unconditionally
                   192    ;
0027               193    SDCMD   EQU     27H             ; Stop gathering frames after GDCMD,
```

```
LOC  OBJ          LINE    SOURCE

                   194                                  ; is an apostrophe.
                   195    ;
003D               196    WBCMD   EQU     '='           ; Warm boot the 8751
                   197    ;
0026               198    BCMD    EQU     '&'           ; Boot the 8751
                   199    ;
0025               200    GTSCMD  EQU     25H           ; Collect one frame of data,
                   201                                  ; is a percent sign.
                   202    ;
002E               203    ACMD    EQU     '.'           ; Average adjacent frames
                   204    ;
002C               205    DACMD   EQU     ','           ; Don't average adjacent frames
                   206    ;
003A               207    TSCMD   EQU     ':'           ; Collect 5 frames after
                   208    ;                               end of word detection
                   209    ;
003B               210    NTSCMD  EQU     ';'           ; Collect no frames after
                   211                                  ; end of word detection
                   212    ;
                   213    ; Special characters sent by this program
                   214    ;
000A               215    NL      EQU     10            ; Marks end of frame.
                   216    ;
002A               217    EOWM    EQU     '*'           ; End of word marker.
                   218    ;
0024               219    EOF     EQU     '$'           ; End of file marker.
                   220    ;
0025               221    OVFM    EQU     25H           ; External buffer overflow indicator,
                   222                                  ; is a percent sign.
                   223    ;
                   224    ; External Buffer space
                   225    ;
2000               226    XMBEG   EQU     2000H         ; Start of external buffer memory
4000               227    XMEND   EQU     3FFFH+1       ; End of external buffer memory + 1 (8K)
                   228    ;
                   229    ; Start address for the multiplexer
                   230    ;
0000               231    MUXST   EQU     0             ; First channel of MUX
                   232    ;
                   233    ;
                   234    ;
                   235    ;-----------------------------------------------------------------
                   236    ;
                   237    ;
                   238    ;
                   239    ;                       ******* BIT ADDRESS SPACE *******
                   240    ;
                   241    ;
----               242            BSEG
                   243    ;
```

```
LOC  OBJ          LINE     SOURCE

                   244     ;               These are flags which indicate the state of the program.
                   245     ;
                   246     ;
                   247     ;
0000               248     STEN:   DBIT    1               ; Set when start of word is detected
                   249                                     ; reset when end of word is detected
0001               250     BUFR:   DBIT    1               ; Set when data in ext buffer is part
                   251                                     ; of a word
0002               252     INHF:   DBIT    1               ; Set by INH to indicate that
                   253                                     ; it wasnt given an ascii hex digit
0003               254     TIF:    DBIT    1               ; Set when serial transmitter is ready
0004               255     RIF:    DBIT    1               ; Set when serial receiver has a byte
0005               256     HCRDY:  DBIT    1               ; Set when the host computer is
                   257                                     ; ready to receive another frame
0006               258     XMT:    DBIT    1               ; Set when there is data in ext buffer
                   259                                     ; awaiting transmission
0007               260     ST:     DBIT    1               ; Set when 3751 is looking for
                   261                                     ; the start of a word.
0008               262     BF:     DBIT    1               ; Set when we want to boot
0009               263     WBF:    DBIT    1               ; Set when we want to warm boot
000A               264     GTSF:   DBIT    1               ; Set when we want to get one frame
000B               265     DSPF:   DBIT    1               ; Set when we want to stop taking
                   266                                     ; in frames
000C               267     GDF:    DBIT    1               ; Set when we want to collect frames
                   268                                     ; unconditionally.
000D               269     TSAF:   DBIT    1               ; Set when we want to collect 5
                   270                                     ; frames of data after the end of
                   271                                     ; word has been detected.
000E               272     ANAF:   DBIT    1               ; Set when we want to average
                   273                                     ; adjacent frames
000F               274     OVFF:   DBIT    1               ; Set when ext buffer overflow is
                   275                                     ; detected
0010               276     GPAF:   DBIT    1               ; Set when we want to enter the
                   277                                     ; parameter subroutine.
                   278     ;
                   279     ;
                   280     ;
                   281     ;-----------------------------------------------------------------------
                   282     ;
                   283     ;
                   284     ;
                   285     ;               ******* TEMPORARY BUFFERS *******
                   286     ;
                   287     ; TBNEW = Marks the beginning of the temporary buffer for current frame.
                   288     ;
                   289     ; TBOLD = Marks the beginning of the temporary buffer for previous frame.
                   290     ;
                   291     ;
                   292     ;-----------------------------------------------------------------------
                   293     ;
```

```
LOC  OBJ            LINE    SOURCE

                     294    ;                  ******** VARIABLES **********
                     295    ;
                     296    ; SPER  = Sampling period value.  This determines the system sampling
                     297    ;         period.  This value is loaded into timer 0 (2 bytes).
                     298    ;         With a clock frequency of 7.3728 MHz, the following values
                     299    ;         give the following system sampling period:
                     300    ;               SPER=E800 - 10 ms
                     301    ;               SPER=D000 - 15 ms
                     302    ;               SPER=CFFF - 20 ms
                     303    ;
                     304    ; STLTM = Settling time constant for log amp.  Used in a count-
                     305    ;         down loop in order to give the log amp time to settle
                     306    ;         down.  In the current system, the log amp is removed.
                     307    ;         Thus, this should always be set to the lowest value, 01.
                     308    ;
                     309    ; THR1  = Threshold value for band one (62.5Hz).
                     310    ;
                     311    ; THR2  = Threshold value for band two (125Hz).
                     312    ;
                     313    ; THR3  = Threshold value for band three (250Hz).
                     314    ;
                     315    ; THR4  = Threshold value for band four (500Hz).
                     316    ;
                     317    ; THR5  = Threshold value for band five (1000Hz).
                     318    ;
                     319    ; THR6  = Threshold value for band six (2000Hz).
                     320    ;
                     321    ; THR7  = Threshold value for band seven (4000Hz).
                     322    ;
                     323    ; SPCC  = The number of consecutive frames needed, in which all the
                     324    ;         bands are below their respective thresholds, before the end
                     325    ;         of a word is declared.
                     326    ;
                     327    ; TSAM  = The number of consecutive frames needed, in which any one
                     328    ;         band in each frame is above its respective threshold, before
                     329    ;         the beginning of a word is declared.
                     330    ;
                     331    ; ATHR  = The number of bands above their respective thresholds in
                     332    ;         the current frame. Calculated for each frame by the routine
                     333    ;         ATHRR and used in CHECK.
                     334    ;
                     335    ; ATHRC = Above threshold consecutive.  The number of consecutive
                     336    ;         frames in which any one band in each frame is above its
                     337    ;         respective threshold.  This is calculated by CHECK and
                     338    ;         used by DETECT.
                     339    ;
                     340    ; BTHRC = Below threshold consecutive.  The number of consecutive
                     341    ;         frames in which all the bands in each frame are below their
                     342    ;         respective thresholds.  This is calculated by CHECK and used
                     343    ;         by DETECT.
```

```
LOC  OBJ          LINE    SOURCE

                   344    ;
                   345    ; NTSS  = The number of frames to be collected (if TSAF set) after the
                   346    ;         end of a word has been detected.
                   347    ;
                   348    ;
                   349    ;
                   350    ;
                   351    ;-------------------------------------------------------------------------
                   352    ;
                   353    ;
                   354    ;
                   355    ;                    ******* DATA ADDRESS SPACE *******
                   356    ;
                   357    ;
----               358            DSEG    AT 30H              ; Start data addresses at hex 30H
                   359    ;
                   360    ;
0030               361    TBNEW:  DS      BPFZC               ; Current frame
003F               362    TBOLD:  DS      BPFZC               ; Previous frame, used when averaging
                   363                                       ; incoming data
004E               364    SPER:   DS      2                   ; Sampling period value
0050               365    STLTM:  DS      1                   ; Settling time constant for log amp,
                   366                                       ; note the log amp is removed.
0051               367    THR1:   DS      1                   ; Threshold 62.5 Hz band
0052               368    THR2:   DS      1                   ; Threshold 125 Hz band
0053               369    THR3:   DS      1                   ; Threshold 250 Hz band
0054               370    THR4:   DS      1                   ; Threshold 500 Hz band
0055               371    THR5:   DS      1                   ; Threshold 1000 Hz band
0056               372    THR6:   DS      1                   ; Threshold 2000 Hz band
0057               373    THR7:   DS      1                   ; Threshold 4000 Hz band
0058               374    SPCC:   DS      1                   ; End of word silence period count
                   375                                       ; (consecutive)
0059               376    TSAM:   DS      1                   ; Beginning of word above threshold
                   377                                       ; count (consecutive)
005A               378    NTSS:   DS      1                   ; Number of frames to take after
                   379                                       ; end of word detection
005B               380    ATHR:   DS      1                   ; Number of bands above threshold
005C               381    ATHRC:  DS      1                   ; Number of consecutive frames above
                   382                                       ; threshold
005D               383    BTHRC:  DS      1                   ; Number of consecutive frames below
                   384                                       ; threshold
005E               385    STORE:  DS      1                   ; Temporary storage area
005F               386    STACK:  DS      1                   ; Start of stack space
                   387    ;
                   388    ;
                   389    ;
                   390    ;-------------------------------------------------------------------------
                   391    ;
                   392    ;
                   393    ;
```

```
LOC  OBJ            LINE     SOURCE

                     394     ;               ******* REGISTER USAGE (BANK 0) *******
                     395     ;
                     396     ;
                     397     ; IPTR          -Points to the next available space in the external buffer
                     398     ;
                     399     ; R7,R6         -Points to the next character to be pulled out of the external
                     400     ;                buffer(R7=high order byte, R6=low order byte).
                     401     ;
                     402     ; R0            -Pointer into temporary buffer
                     403     ;
                     404     ; R2            -Multiplexer address for the ADC0816
                     405     ;
                     406     ; R1,R5         -Scratch registers
                     407     ;
                     408     ;
                     409     ;       ************ (BANK 1) *********
                     410     ;
                     411     ;
                     412     ; R0,R1         -Scratch registers
                     413     ;
                     414     ;
                     415     ;
                     416     ;=================================================================
                     417     ;
                     418     ;
                     419     ;
                     420     ;               ******* POWER-UP RESET *******
                     421     ;
                     422     ;
----                 423             CSEG
0000                 424             ORG     0                       ; Power-up reset starts here
0000 0201CE          425             JMP     BGN                     ; Jump to start of program
                     426     ;
                     427     ;
                     428     ;
                     429     ;=================================================================
                     430     ;
                     431     ;
                     432     ;
                     433     ;         ******* INTERRUPT SERVICE ROUTINE FOR SERIAL I/O *******
                     434     ;
                     435     ;
                     436     ; Common entry point for both transmit and receive serial interrupts
                     437     ;
0023                 438             ORG     23H                     ; Define by hardware
                     439     ;
0023 020100          440             JMP     INTR                    ; Jump to the interrupt routine
                     441     ;
0100                 442             ORG     100H
                     443     ;
```

```
LOC  OBJ          LINE     SOURCE

0100 C0D0          444    INTR:   PUSH    PSW
0102 C0E0          445            PUSH    ACC
0104 309902        446            JNB     TI,I1                   ; Jump if transmitter interrupt
0107 2186          447            AJMP    I21
                   448    ;
                   449    ;  For receiver interrupt.
                   450    ;
0109 C298          451    I1:     CLR     RI                      ; Clear hardware flag
010B E599          452            MOV     A,SBUF                  ; Get data from serial port
010D B40A0E        453            CJNE    A,#RQCMD,I2             ; Jump if not request command
                   454    ;
                   455    ;  Here the host computer is requesting a frame of data.
                   456    ;  We want to set HCRDY when we have one of the following conditions:
                   457    ;       BUFR set-       means we have some data in the ext buffer
                   458    ;                       ready to transmit, i.e., wanted data.
                   459    ;
                   460    ;       or
                   461    ;
                   462    ;       ST set-         means we are activly looking for a word or are
                   463    ;                       in the middle of collecting frames for a word, so
                   464    ;                       in this section we would want to set HCRDY because
                   465    ;                       the host is waiting for data about the spoken word
                   466    ;                       it said was going to be spoken.
                   467    ;
0110 200705        468            JB      ST,I1A
0113 200102        469            JB      BUFR,I1A
0116 21C9          470            AJMP    I20
0118 D205          471    I1A:    SETB    HCRDY
011A 9157          472            ACALL   MAYBE                   ; See if we can send some data
011C 21C9          473            AJMP    I20
                   474    ;
011E B42106        475    I2:     CJNE    A,#STCMD,I3
                   476    ;
                   477    ;  Host says start looking for a word
                   478    ;
0121 713B          479            ACALL   INIP                    ;Initialize pointers
0123 D207          480            SETB    ST
0125 21C9          481            AJMP    I20
                   482    ;
0127 B42309        483    I3:     CJNE    A,#PCMD,I4
                   484    ;
                   485    ;  Here the host wants to change the program parameters
                   486    ;
012A 300702        487            JNB     ST,I3A                  ; Not while looking for word
012D 21C9          488            AJMP    I20
012F D210          489    I3A:    SETB    GPAF                    ; Indicate to main loop to go
                   490                                            ; to parameter routine
0131 21C9          491            AJMP    I20
                   492    ;
0133 B4280D        493    I4:     CJNE    A,#DSPCMD,I5
```

```
LOC  OBJ          LINE     SOURCE

                   494     ;
                   495     ;   Here host wants to stop collecting frames and put an EOF marker
                   496     ;   in ext buffer
                   497     ;
0136 9144          498             ACALL   FILM                    ; Put in file marker
0138 300F04        499             JNB     OVFF,SK1                ; Check for ext buffer overflow
013B 715F          500             ACALL   DET1
013D 21C9          501             AJMP    I20
013F D208          502     SK1:    SETB    DSPF
0141 21C9          503             AJMP    I20
                   504     ;
0143 B42904        505     I5:     CJNE    A,#DSCMD,I6
                   506     ;
                   507     ;   Here we want to start taking in frames of data after being stopped
                   508     ;   by DSPCMD command
                   509     ;
0146 C20B          510             CLR     DSPF
0148 21C9          511             AJMP    I20
                   512     ;
014A B4220C        513     I6:     CJNE    A,#GDCMD,I7
                   514     ;
                   515     ;   Collect frames until overflow or until stopped
                   516     ;
014D 200707        517             JB      ST,I6A                  ; Not while looking for word
0150 200104        518             JB      BUFR,I6A
0153 713D          519             ACALL   INIP                    ; Initialize temp buffer
0155 D20C          520             SETB    GDF
0157 21C9          521     I6A:    AJMP    I20
                   522     ;
0159 B42704        523     I7:     CJNE    A,#SDCMD,I8
                   524     ;
                   525     ;   Stop the collection of frames that was initiated by GDCMD
                   526     ;
015C C20C          527             CLR     GDF
015E 21C9          528             AJMP    I20
                   529     ;
0160 B43D04        530     I8:     CJNE    A,#WBCMD,I9             ;****Warm boot****
0163 D209          531             SETB    WBF
0165 21C9          532             AJMP    I20
                   533     ;
0167 B42604        534     I9:     CJNE    A,#BCMD,I10             ;****Boot command****
016A D208          535             SETB    BF
016C 21C9          536             AJMP    I20
                   537     ;
016E B4250D        538     I10:    CJNE    A,#GTSCMD,I11
                   539     ;
                   540     ;   Here we want to give one frame to the host
                   541     ;
0171 200708        542             JB      ST,I10A                 ; Not if looking for word
0174 200105        543             JB      BUFR,I10A               ; Not if we have valid data
```

```
LOC  OBJ          LINE     SOURCE

0177 200B02        544            JB      DSPF,I10A                ; Not if data input stopped
017A D20A          545            SETB    GTSF
017C 21C9          546     I10A:  AJMP    I20
                   547     ;
017E B42E0A        548     I11:   CJNE    A,#ACMD,I12
                   549     ;
                   550     ;   Here we want to average adjacent frames
                   551     ;
0181 200705        552            JB      ST,I11A                  ; Don't change in mid-word
0184 200102        553            JB      BUFR,I11A                ; Don't change if we have data
0187 D20E          554            SETB    ANAF
0189 21C9          555     I11A:  AJMP    I20
                   556     ;
018B B42C0A        557     I12:   CJNE    A,#DACMD,I13
                   558     ;
                   559     ;    Here we don't want to average adjacent frames
                   560     ;
018E 200705        561            JB      ST,I12A                  ; Don't change in mid-word
0191 200102        562            JB      BUFR,I12A
0194 C20E          563            CLR     ANAF
0196 21C9          564     I12A:  AJMP    I20
                   565     ;
0198 B43A0A        566     I13:   CJNE    A,#TSCMD,I14
                   567     ;
                   568     ;   Collect # NTSS frames after end of word detection
                   569     ;
019B 200705        570            JB      ST,I13A                  ; Don't change in mid-word
019E 200102        571            JB      BUFR,I13A
01A1 D20D          572            SETB    TSAF
01A3 21C9          573     I13A:  AJMP    I20
                   574     ;
01A5 B43B0A        575     I14:   CJNE    A,#NTSCMD,I15
                   576     ;
                   577     ;   Don't collect any frames after end of word detection
                   578     ;
01A8 200705        579            JB      ST,I14A                  ; Don't change in mid-word
01AB 200102        580            JB      BUFR,I14A
01AE C20D          581            CLR     TSAF
01B0 21C9          582     I14A:  AJMP    I20
                   583     ;
01B2 D204          584     I15:   SETB    RIF                      ; Indicate we have received
                   585                                             ; a data byte
01B4 21C9          586            AJMP    I20
                   587     ;
                   588     ;
                   589     ;   For serial transmitter interrupt
                   590     ;
                   591     ;
01B6 C299          592     I21:   CLR     TI                       ; Clear transmitter int.
01B8 D203          593            SETB    TIF                      ; Indicate transmitter ready
```

```
LOC  OBJ          LINE     SOURCE

01BA 30050C         594           JNB     HCRDY,I20                ; Ret if host not ready
01BD 300609         595           JNB     XMT,I20                  ; Ret if nothing to trans.
01C0 91A9           596           ACALL   GETOUT                   ; A has data to output
01C2 9168           597           ACALL   DATAOUT                  ; Transmit the data
01C4 B40A02         598           CJNE    A,#NL,I20                ; Jump if not end data frame
01C7 C205           599           CLR     HCRDY                    ; Host no longer ready
                    600     ;
                    601     ;  Return from the interrupt routine
                    602     ;
01C9 D0E0           603     I20:  POP     ACC
01CB D0D0           604           POP     PSW
01CD 32             605           RETI
                    606     ;
                    607     ;
                    608     ;
                    609     ;-----------------------------------------------------------------
                    610     ;
                    611     ;
                    612     ;
                    613     ;              ******* INITIALIZATION *******
                    614     ;
                    615     ;
                    616     ; Initializes the special registers for serial I/O operation.
                    617     ; as well as the buffer pointers.  Also, the program flags, stack
                    618     ; and internal variables are initialized.
                    619     ;
                    620     ;
01CE 75815F         621     BGN:  MOV     SP,#STACK                ; Initialize stack pointer
                    622     ;
                    623     ; Configure timers 0 and 1:
                    624     ;       timer 0 -> 16 bit counter, sample rate.
                    625     ;       timer 1 -> 8 bit auto-reload, baud rate.
                    626     ;
                    627     ;
01D1 758921         628           MOV     TMOD,#00100001B          ; Set up timers
                    629     ;
                    630     ; Set up serial port clock rate
                    631     ;
01D4 758DFE         632           MOV     TH1,#0FEH                ; Baud rate = 9600
01D7 758BFE         633           MOV     TL1,#0FEH                ;
01DA 758840         634           MOV     TCON,#40H                ; Start timer 1
                    635     ;
                    636     ; Set up serial port mode (8-bit UART)
                    637     ;
01DD 759850         638           MOV     SCON,#01010000B          ; => mode 1, enable serial reception
                    639                                            ; Serial port mode 1
                    640     ;
                    641     ; Enable interrupts from serial port only
                    642     ;
01E0 75A890         643           MOV     IE,#90H                  ; Enable interrupt from serial port
```

```
LOC  OBJ          LINE     SOURCE

01E3 75B810        644            MOV     IP,#10H                 ; Serial port has high priority
                   645      ;
                   646      ; Initialize the program internal parameters to default values.
                   647      ; Boot of program starts here.
                   648      ;
01E6 755A05        649      BOOT:  MOV     NTSS,#5H
01E9 754EE8        650            MOV     SPER,#0E8H              ; System sample period of 10ms,
01EC 754F00        651            MOV     SPER+1,#00H             ;   provided the clock is 7.3728MHz
                   652                                            ;   SPER=DC00 for 15ms
                   653                                            ;   SPER=CFFF for 20ms
01EF 755001        654            MOV     STLTM,#01H              ; Log amp settling time
01F2 75510A        655            MOV     THR1,#0AH               ; Threshold default values
01F5 75520A        656            MOV     THR2,#0AH
01F8 75530A        657            MOV     THR3,#0AH
01FB 75540A        658            MOV     THR4,#0AH
01FE 75550A        659            MOV     THR5,#0AH
0201 75560A        660            MOV     THR6,#0AH
0204 755703        661            MOV     THR7,#03H
0207 75580A        662            MOV     SPCC,#0AH               ; Boundry detection variables
020A 755903        663            MOV     TSAM,#3H
                   664      ;
                   665      ;   Initialize program command flags
                   666      ;
020D C20E          667            CLR     ANAF                    ; No averaging of adjacent frames
020F C20D          668            CLR     TSAF                    ; No frames after end of word
                   669                                            ; detection
                   670      ;
                   671      ; Initialize the flags and the stack.  Warm boot starts here.
                   672      ;
0211 75815F        673      WBOOT: MOV     SP,#STACK
0214 C208          674            CLR     BF                      ; Clear boot flag
0216 C209          675            CLR     WBF                     ; Clear warm boot flag
0218 C205          676            CLR     HCRDY                   ; Host computer not ready
021A C206          677            CLR     XMT                     ; No data are in the buffer
021C C202          678            CLR     INHF                    ; Input byte is ascii hex digit
021E C20F          679            CLR     OVFF                    ; No overflow of ext buffer
0220 C200          680            CLR     STEN                    ; Indicate end of word
0222 C201          681            CLR     BUFR                    ; No word data in ext buffer
0224 C207          682            CLR     ST                      ; Don't look for a start of word
0226 D203          683            SETB    TIF                     ; Transmitter initially ready
0228 C20B          684            CLR     DSPF                    ; Don't stop the input of data
022A C20C          685            CLR     GDF                     ; Don't get data blindly
022C C20A          686            CLR     GTSF                    ; Don't get one frame
022E C204          687            CLR     RIF                     ; Receiver has no data
0230 C210          688            CLR     GPAF                    ; Don't go to the parameter routine
0232 7127          689            ACALL   INIZC                   ; Init ZC counters
                   690      ;
                   691      ;
                   692      ;   Initialize program count variables and temporary buffer STBOLD.
                   693      ;
```

```
LOC  OBJ          LINE     SOURCE

0234 755C00        694            MOV     ATHRC,#0              ; Initialize above threshold
                   695                                          ; counter
0237 755D00        696            MOV     BTHRC,#0              ; Initialize below threshold
                   697                                          ; counter
023A 713D          698            ACALL   INIP                  ; Init. TBOLD temporary buffer
                   699                                          ; and ext buffer pointers
                   700    ;
                   701    ;
                   702    ;-------------------------------------------------------------------------
                   703    ;
                   704    ;
                   705    ;
                   706    ;                       *******   MAIN LOOP  *******
                   707    ;
                   708    ;
                   709    ; This is the main loop. At every system sample time, all the PD's
                   710    ; and ZC's are scanned and  placed into the temporary buffer.
                   711    ; The ZC counters count 1/2 the actual number of zero crossings.
                   712    ; Also, the first ZC values collected for each data group will
                   713    ; always be zero, by nature of the program.  Note, this first frame
                   714    ; will probably not be included as part of a word.  First step in this main
                   715    ; loop is to check the flags to see the state of the program, so that
                   716    ; appropriate actions can be taken.
                   717    ;
023C 200BFD        718    LOOP0:  JB      DSPF,LOOP0            ; Suspend data input.
023F 200713        719            JB      ST,SKIP0              ; Wait until host computer says
                   720                                          ; to look for a word.
0242 2009A1        721            JB      BF,BOOT               ; Boot the program.
0245 2009C9        722            JB      WBF,WBOOT             ; Warm boot the program.
0248 200A0A        723            JB      GTSF,SKIP0            ; Get one frame of data.
024B 200C07        724            JB      GDF,SKIP0             ; Get data unconditionally.
024E 3010EB        725            JNB     GPAF,LOOP0            ; Don't jump means to go into the
                   726                                          ;   PARAM routine.
0251 911D9         727            ACALL   PARAM                 ; Go to PARAM (parameter) routine.
0253 80E7          728            JMP     LOOP0
                   729    ;
                   730    ; Set up timer for the system sampling period
                   731    ;
0255 C28C          732    SKIP0:  CLR     TR0                   ; Disable timer 0 while it's being reset
0257 C28D          733            CLR     TF0                   ; Clear timer 0 overflow
0259 854E8C        734            MOV     TH0,SPER              ; Load timer 0 so it overflows after
025C 854F8A        735            MOV     TL0,SPER+1            ; one sampling period
025F 51EE          736            ACALL   CNTR                  ; Read ZC counter values
0261 D28C          737            SETB    TR0                   ; Start timer 0
0263 7114          738            ACALL   STZC                  ; Start ZC counters
                   739    ;
                   740    ; Initialize for outer loop, the collection of each frame.
                   741    ;
                   742    ;
0265 7830          743            MOV     R0,#TBNEW             ; Initialize temporary buffer pointer
```

```
LOC  OBJ            LINE     SOURCE

0267 7A00            744            MOV     R2,#MUXST               ; Initialize MUX address for 0816
                     745      ;
                     746      ; Inner loop to read each peak detector.
                     747      ;
0269 8A90            748      LOOP1: MOV     P1,R2                   ; Send MUX ADDRESS
026B D2B3            749            SETB    P3.3                    ; input MUX address
026D AD50            750            MOV     R5,STLTM                ; Log amp settling
026F DDFE            751            DJNZ    R5,$                    ;   Note, log amp omitted in final
                     752                                            ;   implementation.
0271 C2B3            753            CLR     P3.3                    ; Start conversion
0273 7590FF          754            MOV     P1,#0FFH                ; Configure Port 1 for input
0276 D2B2            755            SETB    P3.2                    ; Configure P3.2 is input
0278 30B2FD          756            JNB     P3.2,$                  ; Wait until end of conversion
027B D2B4            757            SETB    P3.4                    ; Enable output from A/D
027D E590            758            MOV     A,P1                    ; Read data from pins to Acc
027F C2B4            759            CLR     P3.4                    ; Disable output from A/D
0281 F6              760            MOV     @R0,A                   ; Store data into temporary buffer
0282 0A              761            INC     R2                      ; Next PD channel
0283 08              762            INC     R0                      ; Next temporary buffer location.
0284 BA07E2          763            CJNE    R2,#BPF,LOOP1           ; Repeat until all PD's are sampled
                     764      ;
                     765      ; Call the DETECT routine to decide if we are at the beginning, end or
                     766      ; in the middle of a word.  Also, if we have some valid data, see if we
                     767      ; can send some of that to the host computer.  DETECT will also save the
                     768      ; frame in the ext buffer.
                     769      ;
0287 7153            770            ACALL   DETECT                  ; Word or no word??
0289 C20A            771            CLR     GTSF                    ; Clear flag after getting one time
                     772                                            ; spectral sample, if that is what you
                     773                                            ; are doing.
028B 308DFD          774            JNB     TF0,$                   ; Wait for timer overflow so that
                     775                                            ;   the system sample rate is
                     776                                            ;   consistent.
028E 413C            777            AJMP    LOOP0                   ;
                     778      ;
                     779      ;
                     780      ;
                     781      ;-----------------------------------------------------------------
                     782      ;
                     783      ;
                     784      ;                  ****** SMLP ********
                     785      ;
                     786      ;       This is a second main loop, exactly like the first main loop.
                     787      ;       However, it does not call DETECT and it only executes for #NTSS
                     788      ;       times.  In fact, DETECT calls this main loop in order to collect
                     789      ;       a few time spectral samples after the end of word has been detected.
                     790      ;       This loop is only used when the TSCMD command has been given.
                     791      ;
0290 308DFD          792      SMLP:  JNB     TF0,$                   ; Wait for timer on last sample
                     793                                            ; to finish.
```

```
LOC  OBJ          LINE     SOURCE

0293 AC5A          794            MOV     R4,NTSS             ; Counter for num of samples
0295 300802        795     SMLP1: JNB     BF,SMLP2
0298 21E6          796            AJMP    BOOT                ; Jump to boot routine
029A 300902        797     SMLP2: JNB     WBF,SMLP3
029D 4111          798            AJMP    WBOOT               ; Jump to warm boot routine
029F C28C          799     SMLP3: CLR     TR0                 ; Stop the timer
02A1 C28D          800            CLR     TF0                 ; Clear timer overflow flag
02A3 854E8C        801            MOV     TH0,SPER            ; Initialize counter
02A6 854F8A        802            MOV     TL0,SPER+1
02A9 51EE          803            ACALL   CNTR                ; Read ZC values and initialize
02AB D28C          804            SETB    TR0
02AD 7114          805            ACALL   STZC                ; Start ZC counters
                   806     ;
                   807     ;  Initialize the temporary buffer
                   808     ;
02AF 7830          809            MOV     R0,#TBNEW
02B1 7A00          810            MOV     R2,#MUXST
                   811     ;
                   812     ;  Loop to read all the PD values
                   813     ;
02B3 8A90          814     SMLP4: MOV     P1,R2
02B5 D2B3          815            SETB    P3.3
02B7 AD50          816            MOV     R5,STLTM
02B9 DDFE          817            DJNZ    R5,$
02BB C2B3          818            CLR     P3.3                ; Start the conversion
02BD 7590FF        819            MOV     P1,#0FFH            ; Port 1 for input
02C0 D2B2          820            SETB    P3.2                ; P3.2 is for input
02C2 30B2FD        821            JNB     P3.2,$              ; Wait for end of conversion
02C5 D2B4          822            SETB    P3.4                ; Enable output from ADC
02C7 E590          823            MOV     A,P1                ; Read in the data
02C9 C2B4          824            CLR     P3.4                ; Disable ADC output
02CB F6            825            MOV     @R0,A               ; Save data in temp. buffer
02CC 0A            826            INC     R2                  ; Next MUX address
02CD 08            827            INC     R0                  ; Next temp buffer address
02CE BA07E2        828            CJNE    R2,#BPF,SMLP4       ; Loop until we get all bands
                   829     ;
02D1 9104          830            ACALL   SEND                ; Put data in the ext buffer
02D3 300F02        831            JNB     OVFF,SMLP5
02D6 41DE          832            AJMP    SMLP6               ; Jump if we have ext buffer overflow
02D8 308DFD        833     SMLP5: JNB     TF0,$               ; Wait for timer to overflow
02DB DCB8          834            DJNZ    R4,SMLP1            ; Jump to get rest of the frames
02DD 22            835            RET
                   836     ;
                   837     ;  Here we take care of any overflows that might have occurred when putting
                   838     ;  in the frames after the end of word detection.
                   839     ;
02DE C20F          840     SMLP6: CLR     OVFF
02E0 C206          841            CLR     XMT                 ; Fool interrupt routine so that we
                   842                                        ; can send a character.
02E2 7425          843            MOV     A,#OVFM
```

```
LOC  OBJ            LINE    SOURCE

02E4 3003FD          844             JNB     TIF,$
02E7 9168            845             ACALL   DATAOUT
02E9 D206            846             SETB    XMT
02EB 9131            847             ACALL   WORM                    ; If OVFC routine didn't wipe out the
                     848                                             ; end of word marker, there will be
                     849                                             ; two of them in the ext. buffer.
02ED 22              850             RET
                     851     ;
                     852     ;
                     853     ;
                     854     ;-----------------------------------------------------------------------
                     855     ;
                     856     ;                       CNTR
                     857     ;
                     858     ;       This routine stops the zero crossing (ZC) counters, reads them
                     859     ;       and then stores the ZC values into the temp. buffer @TBNEW. The
                     860     ;       routine then reinitializes the ZC counters back to zero.
                     861     ;       Routine uses R3,R0.
                     862     ;
                     863     ;
02EE C0E0            864     CNTR:   PUSH    ACC
02F0 C083            865             PUSH    DPH
02F2 C082            866             PUSH    DPL
02F4 908000          867             MOV     DPTR,#8000H             ; Address of ZC control register
02F7 7408            868             MOV     A,#08H
02F9 F0              869             MOVX    @DPTR,A                 ; Stop counters
02FA 7837            870             MOV     R0,#TBNEW+7             ; Init. temp. buffer pointer
02FC 7418            871             MOV     A,#18H
02FE F0              872     CNT1:   MOVX    @DPTR,A                 ; Enable ZC value onto P1 control bus
02FF CB              873             XCH     A,R3                    ; R3 has ZC command byte
0300 E590            874             MOV     A,P1                    ; A has ZC value
0302 F6              875             MOV     @R0,A                   ; Save ZC value in temp buffer
0303 CB              876             XCH     A,R3                    ; A has ZC command byte
0304 05E0            877             INC     ACC                     ; Next ZC counter address
0306 08              878             INC     R0                      ; Next temp buffer location
0307 B420F4          879             CJNE    A,#20H,CNT1             ; Jump if we don't have all ZC values
030A 7428            880             MOV     A,#28H
030C F0              881             MOVX    @DPTR,A                 ; Disable and init. ZC counters
030D D082            882             POP     DPL
030F D083            883             POP     DPH
0311 D0E0            884             POP     ACC
0313 22              885             RET
                     886     ;
                     887     ;
                     888     ;-----------------------------------------------------------------------
                     889     ;
                     890     ;
                     891     ;                               STZC
                     892     ;
                     893     ;       This subroutine starts the ZC counters
```

```
LOC  OBJ            LINE     SOURCE

                     894     ;
0314 C0E0            895     STZC:   PUSH    ACC
0316 C082            896             PUSH    DPL
0318 C083            897             PUSH    DPH
031A 7400            898             MOV     A,#00H                  ; Control byte for ZC counter
031C 908000          899             MOV     DPTR,#8000H
031F F0              900             MOVX    @DPTR,A                 ; Start ZC counters
0320 D083            901             POP     DPH
0322 D082            902             POP     DPL
0324 D0E0            903             POP     ACC
0326 22              904             RET
                     905     ;
                     906     ;
                     907     ;-----------------------------------------------------------------
                     908     ;
                     909     ;
                     910     ;                               INIZC
                     911     ;
                     912     ;       This routine stops the ZC counters and initializes them to zero.
                     913     ;
0327 C0E0            914     INIZC:  PUSH    ACC
0329 C083            915             PUSH    DPH
032B C082            916             PUSH    DPL
032D 7428            917             MOV     A,#28H                  ; ZC control byte
032F 908000          918             MOV     DPTR,#8000H             ; ZC control resester address
0332 F0              919             MOVX    @DPTR,A                 ; Stop ZC counters, clear them
0333 7408            920             MOV     A,#08                   ; Another ZC control byte
0335 F0              921             MOVX    @DPTR,A                 ; Keep ZC counters stopped
0336 D082            922             POP     DPL
0338 D083            923             POP     DPH
033A D0E0            924             POP     ACC
033C 22              925             RET
                     926     ;
                     927     ;
                     928     ;-----------------------------------------------------------------
                     929     ;
                     930     ;                       *******   INIP   *******
                     931     ;
                     932     ;
                     933     ;       This subroutine initializes the fifteen bytes temporary buffer
                     934     ;       at TBOLD to all zeros. This is temporary buffer which holds
                     935     ;       the next-to-last frame. Note, the frames are averaged (if ANAF set)
                     936     ;       together, and since the temporary buffer at TBOLD is initially
                     937     ;       zero, the first frame collected will be one-half of the original
                     938     ;       value.  This routine also initializes the put and get pointers
                     939     ;       for the external buffer (DPTR and R7,R6).  Note that if you only want
                     940     ;       to zero out the buffer @TBOLD, you can call part of this routine,
                     941     ;       ITBOLD.
                     942     ;
                     943     ;
```

```
LOC  OBJ          LINE     SOURCE

033D 902000        944     INIP:   MOV     DPTR,#XMBEG                     ; Put pointer of ext buffer
0340 C2D3          945             CLR     RS0                             ; Make sure reg bank 0
0342 7F20          946             MOV     R7,#(HIGH XMBEG)                ; Get pointer (R7=high order)
0344 7E00          947             MOV     R6,#(LOW XMBEG)                 ;             (R6=low order)
0346 C0E0          948     ITBOLD: PUSH    ACC
0348 783F          949             MOV     R0,#TBOLD                       ; Pointer to temp buffer
                   950     ;
                   951     ;  Loop here and zero out the temporary buffer @TBOLD
                   952     ;
034A 7600          953     ITB1:   MOV     @R0,#00H
034C 08            954             INC     R0
034D B83EFA        955             CJNE    R0,#TBOLD+BPF2C,ITB1
0350 D0E0          956             POP     ACC
0352 22            957             RET
                   958     ;
                   959     ;
                   960     ;
                   961     ;-----------------------------------------------------------------
                   962     ;
                   963     ;
                   964     ;
                   965     ;       ******** DETECT *********
                   966     ;
                   967     ;
                   968     ;       This subroutine finds the beginning and end of a word. It
                   969     ;       also keeps track of when we are in the middle of a word.
                   970     ;       Beginning of word = when any one of the seven PD's goes
                   971     ;                           above its respective threshold for #
                   972     ;                           TSAM consecutive frames.
                   973     ;       End of word = when all the PD's go below their respective
                   974     ;                     threshold for # SPCC consecutive frames.
                   975     ;
                   976     ;       This subroutine controls flags ST and STEN (and sets BUFR).
                   977     ;       ST :  indicates that the host wants the LIS program to look
                   978     ;             for a word.
                   979     ;       This subroutine resets ST upon detection of the end of a word.
                   980     ;       STEN  :  set when the beginning of a word has been found.
                   981     ;       ATHRC :  a variable that keeps track of the number of consecutive
                   982     ;                times any PD goes above its respective threshold.
                   983     ;       BTHRC :  a variable that keeps track of the number of consecutive
                   984     ;                times all the PD's go below their thresholds.
                   985     ;
                   986     ;
0353 C0E0          987     DETECT: PUSH    ACC                             ; Save accumulator
0355 200C59        988             JB      GDF,DET6                        ; Skip the detection routine
                   989                                                     ; if we just want to collect
                   990                                                     ; frames
0358 200A56        991             JB      GTSF,DET6                       ; Skip detection routine if
                   992                                                     ; we just want one frame
035B 71BB          993             ACALL   ATHRR                           ; Find out how many bands are
```

```
LOC  OBJ          LINE     SOURCE

                   994                                          ; above and below their
                   995                                          ; thresholds.
                   996     ;
035D 710F          997             ACALL   CHECK                ; Update ATHRC and BTHRC
                   998                                          ; variables, convert and save
                   999                                          ; the frame in ext buffer
                  1000     ;
035F 300F1C        1001     DET1:  JNB     OVFF,DET2
                  1002     ;
                  1003     ;   Here an external buffer overflow has been detected
                  1004     ;
0362 C20C          1005             CLR     GDF                 ; If getting data, stop on
                  1006                                          ; overflow
0364 C20F          1007             CLR     OVFF
0366 C206          1008             CLR     XMT                 ; Fool interrupt routine so
                  1009                                          ; that we can send a char.
0368 7425          1010             MOV     A,#OVFM
036A 3003FD        1011             JNB     TIF,$
036D 9168          1012             ACALL   DATAOUT             ; Send overflow character
036F D206          1013             SETB    XMT
                  1014     ;
                  1015     ;   Now we create an end of word condition
                  1016     ;
0371 C200          1017             CLR     STEN                ; Indicate end of word found
0373 C207          1018             CLR     ST                  ; No longer looking for a word.
0375 9131          1019             ACALL   WORM                ; End of word marker
0377 9144          1020             ACALL   FILM                ; End of file marker
0379 D0E0          1021             POP     ACC
037B 7127          1022             ACALL   INIZC               ; Initialize ZC counters
037D 22            1023             RET
                  1024     ;
037E 20000A        1025     DET2:  JB      STEN,IWORD           ; If in word no need to look
                  1026                                          ; for beginning of word
0381 C3            1027             CLR     C
0382 E55C          1028             MOV     A,ATHRC
0384 9559          1029             SUBB    A,TSAM
0386 600D          1030             JZ      BWORD               ; If ACOUNT = # TSAM, then
0388 300011        1031             JNB     STEN,NWORD          ; beginning of a word
038B C3            1032     IWORD: CLR     C
038C E55D          1033             MOV     A,BTHRC             ; BTHRC= SPCC, then end word
038E 9558          1034             SUBB    A,SPCC
0390 600B          1035             JZ      EWORD               ; If not zero, then it is the
0392 D0E0          1036             POP     ACC                 ; middle of a word.
0394 22            1037             RET
                  1038     ;
                  1039     ; Here, the beginning of a word is detected.
                  1040     ; At least one of the bands was above its threshold for TSAM consecutive
                  1041     ; frames. Note that it doesn't have to be the same band in each
                  1042     ; of the frames.
                  1043     ;
```

```
LOC  OBJ            LINE     SOURCE

0395 D200           1044     EWORD:  SETB    STEN                    ; STEN is set when ATHRC=TSAM
0397 D201           1045             SETB    BUFR                    ; Indicate we have wanted data
                    1046                                             ; in the ext. buffer
                    1047
0399 D0E0           1048             POP     ACC
039B 22             1049             RET
                    1050     ;
                    1051     ; Here, no word is detected.
                    1052     ;
039C D0E0           1053     NWORD:  POP     ACC
039E 22             1054             RET
                    1055     ;
                    1056     ; Here, the end of a word is detected.
                    1057     ;
039F C200           1058     EWORD:  CLR     STEN                    ; STEN is reset when
                    1059                                             ; BTHRC=SPCC
03A1 C207           1060             CLR     ST                      ; Not looking for a word
03A3 9131           1061             ACALL   WORM
03A5 300D02         1062             JNB     TSAF,DET5               ; Jump if we don't want frames
                    1063                                             ; after end of word detection
03A8 5190           1064             ACALL   SMLP                    ; Get the extra frames
03AA 9144           1065     DET5:   ACALL   FILM                    ; Put EOF marker in ext buffer
03AC 7127           1066             ACALL   INIZC                   ; Initialize ZC counters
03AE D0E0           1067             POP     ACC
03B0 22             1068             RET
                    1069     ;
                    1070     ;  Execute this when skipping this routine i.e., when getting data blindly
                    1071     ;  or when getting one frame.
                    1072     ;
03B1 9104           1073     DET6:   ACALL   SEND
03B3 D201           1074             SETB    BUFR                    ; Indicate we have wanted data
                    1075                                             ; in the ext. buffer
03B5 200FA7         1076             JB      OVFF,DET1
03B8 D0E0           1077             POP     ACC
03BA 22             1078             RET
                    1079     ;
                    1080     ;
                    1081     ;
                    1082     ;-----------------------------------------------------------------------
                    1083     ;
                    1084     ;
                    1085     ;                    ******* ATHRR *******
                    1086     ;
                    1087     ;
                    1088     ;       This routine goes through the PD values that are stored in the
                    1089     ;       temporary buffer @ TBNEW. It sets ATRR to equal the number
                    1090     ;       of bands that are above their respective thresholds.
                    1091     ;       This routine doesn't destroy any registers and needs no
                    1092     ;       parameters passed to it.
                    1093     ;
```

```
LOC  OBJ          LINE    SOURCE

                  1094    ;
                  1095    ;
03BB C0E0         1096    ATHRR:  PUSH    ACC
03BD E8           1097            MOV     A,R0
03BE C0E0         1098            PUSH    ACC
03C0 E9           1099            MOV     A,R1
03C1 C0E0         1100            PUSH    ACC
03C3 755B00       1101            MOV     ATHR,#0                 ; Initialize above thr count
03C6 7830         1102            MOV     R0,#TBNEW               ; Temp buffer pointer
03C8 7951         1103            MOV     R1,#THR1                ; Threshold value pointer
                  1104    ;
03CA E6           1105    ATH2:   MOV     A,@R0
03CB C3           1106            CLR     C
03CC 97           1107            SUBB    A,@R1
03CD 4002         1108            JC      ATH1                    ; Jump if below threshold
03CF 055B         1109            INC     ATHR
03D1 08           1110    ATH1:   INC     R0
03D2 09           1111            INC     R1
03D3 B958F4       1112            CJNE    R1,#THR7+1,ATH2         ; Jump until last band is
                  1113                                            ; compared with its thr
03D6 D0E0         1114            POP     ACC                     ; Restore registers
03D8 F9           1115            MOV     R1,A
03D9 D0E0         1116            POP     ACC
03DB F8           1117            MOV     R0,A
03DC D0E0         1118            POP     ACC
03DE 22           1119            RET
                  1120    ;
                  1121    ;
                  1122    ;
                  1123    ;
                  1124    ;-----------------------------------------------------------------
                  1125    ;
                  1126    ;
                  1127    ;
                  1128    ;                       *******  CHECK  *******
                  1129    ;
                  1130    ;
                  1131    ;       This subroutine uses ATHR (which is calculated in ATHRR) to
                  1132    ;       update ATHRC and BTHRC.
                  1133    ;       ATHRC  = number of consecutive frames any one band goes above its
                  1134    ;                threshold.
                  1135    ;       BTHRC  = number of consecutive frames all the bands go below their
                  1136    ;                threshold.
                  1137    ;       This subroutine also detects 'false data' and then reinitializes
                  1138    ;       ext buffer to get rid of this.  False data occurs when the consecutive
                  1139    ;       pattern is broken while looking for a word.
                  1140    ;
                  1141    ;
03DF C0E0         1142    CHECK:  PUSH    ACC
03E1 E55B         1143            MOV     A,ATHR
```

```
LOC  OBJ          LINE    SOURCE

03E3 C3           1144            CLR     C
03E4 9400         1145            SUBB    A,#0
03E6 600A         1146            JZ      BELOWT                  ; Jump if all below thr
                  1147    ;
                  1148    ;   Here at least one of the bands is above its threshold
                  1149    ;
03E8 055C         1150            INC     ATHRC                   ; Inc. consecutive count
                  1151    ;
03EA 9104         1152            ACALL   SEND                    ; put data in ext buffer
03EC 755D00       1153            MOV     BTHRC,#0                ; Reinitialize BTHRC because its
                  1154                                            ; consecutive pattern broke
03EF D0E0         1155            POP     ACC
03F1 22           1156            RET
                  1157    ;
                  1158    ; Here all the bands are below their thresholds, and we must find out if
                  1159    ; we are in the middle of a word or if we are still searching for a word.
                  1160    ;
03F2 055D         1161    BELOWT: INC     BTHRC
03F4 755C00       1162            MOV     ATHRC,#0                ; Reinitialize ATHRC because its
                  1163                                            ; consecutive pattern broke
03F7 300005       1164            JNB     STEN,NSTART             ; Jump if not in middle of word
                  1165                                            ; because all bands were below thr
                  1166    ;
03FA 9104         1167            ACALL   SEND                    ; Here we are in the middle of a
                  1168                                            ; word so we convert and store the
                  1169                                            ; frame @TBNEW in the ext. buffer
03FC D0E0         1170            POP     ACC                     ;
03FE 22           1171            RET
                  1172    ;
                  1173    ; Here, all the bands were below their thresholds and we were not
                  1174    ; within a word.  This means everything stored in ext buffer is
                  1175    ; invalid, so reinitialize everything to get rid of these frames.
                  1176    ;
03FF 713D         1177    NSTART: ACALL   INIP                    ; Reinitialize buffer @TBOLD and
                  1178                                            ; reset the ext buffer pointers
0401 D0E0         1179            POP     ACC                     ;
0403 22           1180            RET
                  1181    ;
                  1182    ;
                  1183    ;
                  1184    ;-----------------------------------------------------------------------
                  1185    ;
                  1186    ;
                  1187    ;
                  1188    ;                       ******* SEND *******
                  1189    ;
                  1190    ;
                  1191    ;       This subroutine averages each frame with the previous frame
                  1192    ;       (if ANAF is set), then converts each byte of this averaged frame
                  1193    ;       into two ascii hex bytes and puts it into the ext buffer.
```

```
LOC  OBJ          LINE    SOURCE

                  1194    ;       Each frame is 30 bytes of ascii data in the external buffer,
                  1195    ;       14 bytes of BPF values and 16 bytes of ZC values.
                  1196    ;
                  1197    ;
0404 C0E0         1198    SEND:   PUSH    ACC                     ;
0406 D2D3         1199            SETB    RS0                     ; use register BANK 1
0408 7830         1200            MOV     R0,#TBNEW               ; Current frame
040A 793F         1201            MOV     R1,#TBOLD               ; Previous frame
040C 7A00         1202            MOV     R2,#0                   ;
                  1203    ;
                  1204    ; Loop converts each byte to ascii and then puts this into the ext buffer.
                  1205    ;
040E 20DF1B       1206    SEN1:   JB      OVFF,SEN2               ; Jump if we have overflow condition
0411 E6           1207            MOV     A,@R0                   ; A has one byte of current frame
0412 30DE05       1208            JNB     ANAF,SEN3               ; Skip if we don't want to average
0415 C7           1209            XCH     A,@R1                   ; Stores as previous frame
0416 26           1210            ADD     A,@R0                   ; Average the two bytes
0417 13           1211            RRC     A                       ; Divides by 2
0418 3400         1212            ADDC    A,#0                    ; Round off
041A C2D3         1213    SEN3:   CLR     RS0                     ; back to register bank 0
                  1214    ;
                  1215    ; Put data into ext buffer.
                  1216    ;
041C 916F         1217            ACALL   CONVERT                 ; Data are now ASCII, and in ext buffer
041E D2D3         1218            SETB    RS0                     ; Return to reg. bank 1
0420 08           1219            INC     R0                      ; Inc to next data byte
0421 09           1220            INC     R1
0422 0A           1221            INC     R2
0423 BA0FE8       1222            CJNE    R2,#BPFZC,SEN1          ; Loop until we set all bands
0426 C2D3         1223            CLR     RS0                     ; Make sure in bank 0
0428 740A         1224            MOV     A,#NL
042A 918D         1225            ACALL   PUTIN                   ; Put a line feed in ext buffer
                  1226                                            ; after each frame
042C D0E0         1227    SEN2:   POP     ACC
042E C2D3         1228            CLR     RS0                     ; Return to reg. bank 0
0430 22           1229            RET
                  1230    ;
                  1231    ;
                  1232    ;
                  1233    ;-----------------------------------------------------------------------
                  1234    ;
                  1235    ;
                  1236    ;
                  1237    ;                       ******* WORM *******
                  1238    ;
                  1239    ;
                  1240    ;       This routine puts an end of word marker into the ext buffer
                  1241    ;       followed by a new line (NL). The end of word marker is '*'.
                  1242    ;
                  1243    ;
```

```
LOC  OBJ           LINE     SOURCE

0431 200F0F        1244     WORM:   JB      OVFF,WOR1           ; Skip if in overflow condition
0434 C0E0          1245             PUSH    ACC
0436 742A          1246             MOV     A,#EOWM             ; Put in end of word marker
0438 918B          1247             ACALL   PUTIN
043A 200F06        1248             JB      OVFF,WOR1           ; Jump if ext. buffer is overflowed
043D 740A          1249             MOV     A,#NL               ; Put in a NL character
043F 918D          1250             ACALL   PUTIN
0441 D0E0          1251             POP     ACC
0443 22            1252     WOR1:   RET
                   1253     ;
                   1254     ;
                   1255     ;
                   1256     ;-------------------------------------------------------------------
                   1257     ;
                   1258     ;
                   1259     ;               ********** FILM ***********
                   1260     ;
                   1261     ;
                   1262     ;       This routine puts an end of file marker into the external buffer
                   1263     ;       followed by a new line (NL).   The end of file marker is '$'.
                   1264     ;
                   1265     ;
0444 200F0F        1266     FILM:   JB      OVFF,FIL1           ; Skip if in overflow condition
0447 C0E0          1267             PUSH    ACC
0449 7424          1268             MOV     A,#EOF              ; Put in end of file marker
044B 918D          1269             ACALL   PUTIN
044D 200F06        1270             JB      OVFF,FIL1           ; Jump if overflow
0450 740A          1271             MOV     A,#NL               ; Put in NL character in ext buffer
0452 918D          1272             ACALL   PUTIN
0454 D0E0          1273             POP     ACC
0456 22            1274     FIL1:   RET
                   1275     ;
                   1276     ;
                   1277     ;-------------------------------------------------------------------
                   1278     ;
                   1279     ;
                   1280     ;
                   1281     ;
                   1282     ;
                   1283     ;               ******* MAYBE *******
                   1284     ;
                   1285     ;
                   1286     ;       This routine checks to see if a data byte transmission
                   1287     ;       to the host can be initiated.
                   1288     ;
                   1289     ;
0457 30030D        1290     MAYBE:  JNB     TIF,M1              ; Transmitter ready ?
045A 30050A        1291             JNB     HCRDY,M1            ; Host ready ?
045D 300607        1292             JNB     XMT,M1              ; Data to transmit ?
0460 300104        1293             JNB     BUFR,M1             ; Are the data in the ext buffer part
```

```
LOC  OBJ          LINE     SOURCE

                  1294                                    ; of a word?
0463 91A9         1295           ACALL   GETOUT           ; Yes, get data from ext buffer
0465 9168         1296           ACALL   DATAOUT          ; Send the byte
0467 22           1297     M1:   RET
                  1298     ;
                  1299     ;
                  1300     ;
                  1301     ;-----------------------------------------------------------
                  1302     ;
                  1303     ;
                  1304     ;
                  1305     ;              ******* DATAOUT *******
                  1306     ;
                  1307     ;
                  1308     ;      This routine takes data from the Acc and transmits it through
                  1309     ;      the serial I/O port.
                  1310     ;
0468 C299         1311     DATAOUT:CLR    TI
046A C203         1312           CLR     TIF              ; Transmitter no longer ready
046C F599         1313           MOV     SBUF,A           ; Initiate transmission
046E 22           1314           RET
                  1315     ;
                  1316     ;
                  1317     ;
                  1318     ;-----------------------------------------------------------
                  1319     ;
                  1320     ;
                  1321     ;
                  1322     ;            ******* CONVERT *******
                  1323     ;
                  1324     ;
                  1325     ;      Takes byte in A, converts it into two ascii digits and puts
                  1326     ;      them into the external buffer.  This routine also watches out for
                  1327     ;      overflow conditions by monitoring the OVFF flag and calling the
                  1328     ;      OVFC routine.  Lower order byte is put into the external buffer
                  1329     ;      first and then the higher order byte.
                  1330     ;
                  1331     ;
046F C0E0         1332     CONVERT:PUSH   ACC              ; Save byte
0471 C4           1333           SWAP    A                ; Shift to right 4 bits
0472 9184         1334           ACALL   HEX              ; Convert higher nibble of A
0474 918D         1335           ACALL   PUTIN            ; Put ascii hex digit in ext buffer
0476 200F03       1336           JB      OVFF,CON1        ; Jump if there is an overflow
0479 D0E0         1337           POP     ACC              ; Get original byte
047B C0E0         1338           PUSH    ACC              ; Save it again
047D 9184         1339           ACALL   HEX              ; Convert lower nibble of A
047F 918D         1340           ACALL   PUTIN            ; Put ascii hex digit in ext buffer
0481 D0E0         1341     CON1: POP     ACC              ; Set original byte back
0483 22           1342           RET
                  1343     ;
```

```
LOC  OBJ            LINE    SOURCE

                    1344    ;
                    1345    ;
                    1346    ;-----------------------------------------------------------------------
                    1347    ;
                    1348    ;
                    1349    ;
                    1350    ;                       ******* HEX *******
                    1351    ;
                    1352    ;
                    1353    ;       Converts the lower four bits of A into an ascii hex digit.
                    1354    ;       Returns the value in the ACC.
                    1355    ;
0484 540F           1356    HEX:    ANL     A,#0FH                  ; Retain lower 4 bits
0486 2490           1357            ADD     A,#90H                  ; Convert to ASCII hex digit
0488 D4             1358            DA      A
0489 3440           1359            ADDC    A,#40H
048B D4             1360            DA      A
048C 22             1361            RET
                    1362    ;
                    1363    ;
                    1364    ;
                    1365    ;-----------------------------------------------------------------------
                    1366    ;
                    1367    ;
                    1368    ;
                    1369    ;                       ******* PUTIN *******
                    1370    ;
                    1371    ;
                    1372    ;       Moves byte in ACC to the external buffer. DPTR has the address to which
                    1373    ;       the byte will be moved. This routine also resets the DPTR when it
                    1374    ;       gets to the end of the external buffer.  DPTR is the put pointer.
                    1375    ;
                    1376    ;
048D C2D3           1377    PUTIN:  CLR     RS0                     ; Make sure in reg bank 0
048F F0             1378            MOVX    @DPTR,A                 ; Move the data into ext buffer
0490 D206           1379            SETB    XMT                     ; Something is now in ext buffer
0492 A3             1380            INC     DPTR                    ; Next ext buffer location
0493 C0E0           1381            PUSH    ACC                     ; Save Acc value
0495 E583           1382            MOV     A,DPH                   ; Check that DPTR is still within
0497 B44008         1383            CJNE    A,#(HIGH XMEND),PUT1    ;     limits
049A E582           1384            MOV     A,DPL
049C B40003         1385            CJNE    A,#(LOW XMEND),PUT1
049F 902000         1386            MOV     DPTR,#XMBEG             ; Reset DPTR if outside limits
04A2 B12C           1387    PUT1:   ACALL   OVFC                    ; Check for overflow and set OVFF
04A4 D0E0           1388            POP     ACC                     ; Restore Acc
04A6 9157           1389            ACALL   MAYBE                   ; See if can send something
04A8 22             1390    PUT2:   RET
                    1391    ;
                    1392    ;
                    1393    ;
```

```
LOC  OBJ            LINE   SOURCE

                    1394   ;-----------------------------------------------------------------------
                    1395   ;
                    1396   ;
                    1397   ;
                    1398   ;                       ******* GETOUT *******
                    1399   ;
                    1400   ;
                    1401   ;       Transfers one data byte from the ext buffer to the ACC. Uses R7, R6 as
                    1402   ;       pointer to the data byte.  R7, R6 is the get pointer.
                    1403   ;
                    1404   ;
04A9 C2D3           1405   GETOUT: CLR     RS0                     ; Make sure in reg bank 0
04AB C083           1406           PUSH    DPH
04AD C082           1407           PUSH    DPL                     ; Save DPTR
04AF 8F83           1408           MOV     DPH,R7                  ; Set up DPTR as get pointer
04B1 8E82           1409           MOV     DPL,R6
04B3 E0             1410           MOVX    A,@DPTR                 ; Get data from ext buffer
04B4 A3             1411           INC     DPTR                    ; Increment get pointer
04B5 AF83           1412           MOV     R7,DPH                  ; Load incremented pointer back
04B7 AE82           1413           MOV     R6,DPL                  ; into R7, R6
04B9 D082           1414           POP     DPL
04BB D083           1415           POP     DPH                     ; Restore the put pointer
04BD BF4007         1416           CJNE    R7,#(HIGH XMEND),GET1   ; Check that R7, R6 are within
04C0 BE0004         1417           CJNE    R6,#(LOW XMEND),GET1    ;    limits
04C3 7F20           1418           MOV     R7,#(HIGH XMBEG)        ; Else must reset
04C5 7E00           1419           MOV     R6,#(LOW XMBEG)
04C7 C0E0           1420   GET1:   PUSH    ACC                     ; Save data byte
04C9 EF             1421           MOV     A,R7                    ; See if any more data
04CA B58308         1422           CJNE    A,DPH,GET2              ; No more data when put pointer
04CD EE             1423           MOV     A,R6                    ; equals the get pointer
04CE B58204         1424           CJNE    A,DPL,GET2
04D1 C206           1425           CLR     XMT
04D3 C201           1426           CLR     BUFR                    ; Indicate no more data to transmit
04D5 D0E0           1427   GET2:   POP     ACC                     ; Restore data byte
04D7 22             1428           RET
                    1429   ;
                    1430   ;
                    1431   ;
                    1432   ;-----------------------------------------------------------------------
                    1433   ;
                    1434   ;
                    1435   ;
                    1436   ;                   ************ PARAM ***************
                    1437   ;
                    1438   ;       Routine used to change important program parameters.  When invoked
                    1439   ;       the routine sends out the current values of all the parameters in
                    1440   ;       the following order: SPER, STLTM, THR1, THR2, THR3, THR4, THR5,
                    1441   ;       THR6, THR7, SPCC, TSAM, CR.  The CR indicates the end.  After this,
                    1442   ;       the routine waits for 24 ascii bytes to be converted and used as the
                    1443   ;       new values of these parameters.  Each parameter is 1 byte long (2 ascii
```

```
LOC  OBJ          LINE    SOURCE

                  1444    ;       bytes), except for SPER which is 2 bytes long (4 ascii bytes).  Input
                  1445    ;       is in the same order as that which was displayed.  If a non ascii
                  1446    ;       hex digit, eg CR, is received, then the old value for that parameter
                  1447    ;       is saved.  Note, you must enter the correct number of ascii digits
                  1448    ;       for each parameter whether or not you are saving the old value
                  1449    ;       or not.
                  1450    ;
                  1451    ;
04D3 C210         1452    PARAM:  CLR     GPAF            ; We are honoring the request
04DA C204         1453            CLR     RIF             ; Initialize receiver interrupt flag
04DC B13D         1454            ACALL   DPAR            ; Display parameters, send out serial
                  1455                                    ; port in ascii format
04DE C0E0         1456            PUSH    ACC
04E0 D2D3         1457            SETB    RS0             ; Register bank 1
04E2 B12D         1458            ACALL   REC             ; Get first ascii hex digit for SPER
04E4 FA           1459            MOV     R2,A
04E5 B135         1460            ACALL   TRANS           ; Echo out first digit
04E7 740A         1461            MOV     A,#NL           ; Needed for handshaking on
04E9 B135         1462            ACALL   TRANS           ;   the host
04EB B12D         1463            ACALL   REC             ; Get 2nd ascii byte for SPER
04ED FB           1464            MOV     R3,A
04EE B135         1465            ACALL   TRANS           ; Echo out 2nd digit
04F0 740A         1466            MOV     A,#NL           ; Needed for handshaking on
04F2 B135         1467            ACALL   TRANS           ;   the host
04F4 B11A         1468            ACALL   INPA            ; Input last two ascii hex digits
                  1469                                    ; into R4 and R5
04F6 784E         1470            MOV     R0,#SPER        ; Pointer to SPER
04F8 B161         1471            ACALL   INH             ; Convert for SPER, the only 4
                  1472                                    ; byte ascii hex digit
04FA 200205       1473            JB      INHF,PA1        ; Digit not ascii hex causes jump
04FD EC           1474            MOV     A,R4
04FE F6           1475            MOV     @R0,A           ; Save HO SPER
04FF 08           1476            INC     R0
0500 ED           1477            MOV     A,R5
0501 F6           1478            MOV     @R0,A           ; Save LO SPER
0502 7850         1479    PA1:    MOV     R0,#SPER+2
0504 7A30         1480    PA2:    MOV     R2,#'0'         ; All other parameters are two ascii
                  1481                                    ; hex digits
0506 7B30         1482            MOV     R3,#'0'
0508 B11A         1483            ACALL   INPA            ; Get next parameter
050A B161         1484            ACALL   INH             ; Convert it to hex number
050C 200202       1485            JB      INHF,PA3        ; Jump to save old value
050F ED           1486            MOV     A,R5
0510 F6           1487            MOV     @R0,A           ; Save new parameter value
0511 08           1488    PA3:    INC     R0
0512 B85AEF       1489            CJNE    R0,#SPER+12,PA2 ; Loop until we got all parameters
0515 C2D3         1490            CLR     RS0             ; Back to bank 0
0517 D0E0         1491            POP     ACC
0519 22           1492            RET
                  1493    ;
```

```
LOC  OBJ            LINE     SOURCE

                    1494     ;
                    1495     ;-----------------------------------------------------------------
                    1496     ;
                    1497     ;
                    1498     ;                  ********** INPA **************
                    1499     ;
                    1500     ;
                    1501     ;       This routine waits for two ascii bytes of input from the serial
                    1502     ;       port, and puts these bytes into R4,R5 (R4 has the first byte).
                    1503     ;       Must be in bank 1 before using. Routine used by PARAM.
                    1504     ;
                    1505     ;
051A B12D           1506     INPA:   ACALL   REC                     ; Get byte from serial port into A
051C FC             1507             MOV     R4,A
051D B135           1508             ACALL   TRANS                   ; Echo byte out serial port
051F 740A           1509             MOV     A,#NL                   ; Needed for handshaking with
0521 B135           1510             ACALL   TRANS                   ;   the host
0523 B12D           1511             ACALL   REC                     ; Get 2nd byte from serial port into A
0525 FD             1512             MOV     R5,A
0526 B135           1513             ACALL   TRANS                   ; Echo byte out serial port
0528 740A           1514             MOV     A,#NL                   ; Needed for handshaking with
052A B135           1515             ACALL   TRANS                   ;   the host
052C 22             1516             RET
                    1517     ;
                    1518     ;
                    1519     ;
                    1520     ;-----------------------------------------------------------------
                    1521     ;
                    1522     ;
                    1523     ;
                    1524     ;                  *********** REC ****************
                    1525     ;
                    1526     ;       Routine waits for input from the serial port and puts the byte
                    1527     ;       into A.
                    1528     ;
052D 3004FD         1529     REC:    JNB     RIF,$                   ; Wait till a byte has been received
0530 C204           1530             CLR     RIF
0532 E599           1531             MOV     A,SBUF
0534 22             1532             RET
                    1533     ;
                    1534     ;
                    1535     ;-----------------------------------------------------------------
                    1536     ;
                    1537     ;
                    1538     ;                  ************ TRANS ***************
                    1539     ;
                    1540     ;
                    1541     ;       Routine waits for transmitter to be ready and then sends the byte
                    1542     ;       in A out the serial port.
                    1543     ;
```

```
LOC  OBJ            LINE    SOURCE

                    1544    ;
0535 3003FD         1545    TRANS:  JNB     TIF,$                   ; Wait for transmitter to be ready
0538 C203           1546            CLR     TIF
053A F599           1547            MOV     SBUF,A
053C 22             1548            RET
                    1549    ;
                    1550    ;
                    1551    ;-----------------------------------------------------------------------
                    1552    ;
                    1553    ;
                    1554    ;               ***************** DPAR ******************
                    1555    ;
                    1556    ;
                    1557    ;       This routine sets the parameters, converts them to ascii hex, and
                    1558    ;       then sends them out the serial port for display in the following
                    1559    ;       order: SPER (4 ascii hex digits), STLTM, THR1, THR2, THR3, THR4,
                    1560    ;       THR5, THR6, THR7, SPCC, TSAM, CR.  The CR indicates the end of
                    1561    ;       transmission.  All parameters are 2 ascii hex digits unless other-
                    1562    ;       wise stated.  This routine doesnt destroy any regesters and doesnt
                    1563    ;       need to be in bank 1.
                    1564    ;
                    1565    ;
053D C0E0           1566    DPAR:   PUSH    ACC
053F E8             1567            MOV     A,R0
0540 C0E0           1568            PUSH    ACC
0542 784E           1569            MOV     R0,#SPER                ; Pointer to begin of parameter space
0544 E6             1570    DP1:    MOV     A,@R0
0545 C4             1571            SWAP    A
0546 9184           1572            ACALL   HEX                     ; Convert HO nibble to ascii hex digit
0548 B135           1573            ACALL   TRANS                   ; Send the digit out the serial port
054A E6             1574            MOV     A,@R0
054B 9184           1575            ACALL   HEX                     ; Convert LO nibble to ascii hex digit
054D B135           1576            ACALL   TRANS                   ; Send LO digit out serial port
054F 7420           1577            MOV     A,#SPACE
0551 B135           1578            ACALL   TRANS                   ; Insert space between each two digits
0553 08             1579            INC     R0
0554 B85AED         1580            CJNE    R0,#SPER+12,DP1         ; Jump until all parameters are output
0557 740D           1581            MOV     A,#CR
0559 B135           1582            ACALL   TRANS                   ; Output CR at the end
055B D0E0           1583            POP     ACC
055D F8             1584            MOV     R0,A
055E D0E0           1585            POP     ACC
0560 22             1586            RET
                    1587    ;
                    1588    ;
                    1589    ;
                    1590    ;-----------------------------------------------------------------------
                    1591    ;
                    1592    ;
                    1593    ;               ************ INH ******************
```

```
LOC  OBJ          LINE     SOURCE

                  1594     ;
                  1595     ;
                  1596     ;       This routine is called from PARAM routine, and the 8751 should
                  1597     ;       be using bank 1 when calling this routine, as it uses four registers
                  1598     ;       for parameter passing.  This routine takes four ascii hex digits
                  1599     ;       in R2 (HO), R3, R4, R5 (LO) and converts it into a 16 bit base
                  1600     ;       2 hexadecimal number and puts the result in R4 (HO), R5 (LO).
                  1601     ;       If conversion was successful, INHF is reset, if not successful
                  1602     ;       (byte was not an ascii hex number) INHF is set.  If not successful
                  1603     ;       then R2,R3,R4,R5 contains indeterminant data.
                  1604     ;
                  1605     ;
0561 C0E0         1606     INH:    PUSH    ACC
0563 C202         1607             CLR     INHF                    ; Initially assume conversion ok
0565 ED           1608             MOV     A,R5
0566 B17C         1609             ACALL   INH1                    ; Lower half of A has hex nibble
0568 FD           1610             MOV     R5,A
0569 EC           1611             MOV     A,R4
056A B17C         1612             ACALL   INH1
056C C4           1613             SWAP    A
056D 2D           1614             ADD     A,R5
056E FD           1615             MOV     R5,A                    ; R5 has lower 2 hex digit results
056F EB           1616             MOV     A,R3
0570 B17C         1617             ACALL   INH1
0572 FB           1618             MOV     R3,A
0573 EA           1619             MOV     A,R2
0574 B17C         1620             ACALL   INH1
0576 C4           1621             SWAP    A
0577 2B           1622             ADD     A,R3
0578 FC           1623             MOV     R4,A                    ; R4 has HO two hex digit results
0579 D0E0         1624             POP     ACC
057B 22           1625             RET
                  1626     ;
                  1627     ;
                  1628     ;
                  1629     ;-------------------------------------------------------------------------
                  1630     ;
                  1631     ;
                  1632     ;                  ********** INH1 **************
                  1633     ;
                  1634     ;
                  1635     ;       This routine takes the ascii digit in the Acc and returns a hex
                  1636     ;       nibble in the lower half of the Acc, unless the ascii digit was
                  1637     ;       not an ascii hex digit in which case it sets flag INHF, and
                  1638     ;       returns the ascii digit in the Acc.  This routine does not reset
                  1639     ;       INHF.
                  1640     ;
                  1641     ;
057C F55E         1642     INH1:   MOV     STORE,A
057E FE           1643             MOV     R6,A
```

```
LOC  OBJ          LINE    SOURCE

057F 547F         1644            ANL     A,#7FH                  ; Mask off parity bit
0581 C0E0         1645            PUSH    ACC
0583 C3           1646            CLR     C
0584 9440         1647            SUBB    A,#40H
0586 401E         1648            JC      INH1A                   ; Jump if # between 0-9
0588 D0E0         1649            POP     ACC
                  1650    ;
                  1651    ;   Here the number should be between A-F.
                  1652    ;
058A C3           1653            CLR     C
058B 9437         1654            SUBB    A,#37H                  ; A=hex number between A-F.
                  1655    ;   Check to see if we do have a valid hex number between A-F.
058D C0E0         1656            PUSH    ACC
058F C3           1657            CLR     C
0590 9410         1658            SUBB    A,#10H                  ; Should set the carry.
0592 4002         1659            JC      INH1B                   ; Jump if passed test
0594 D202         1660            SETB    INHF
0596 D0E0         1661    INH1B:  POP     ACC
0598 C0E0         1662            PUSH    ACC
059A C3           1663            CLR     C
059B 940A         1664            SUBB    A,#0AH                  ; Should not set carry
059D 5002         1665            JNC     INH1C                   ; Jump if good number
059F D202         1666            SETB    INHF
05A1 D0E0         1667    INH1C:  POP     ACC
05A3 0205B6       1668            JMP     INH1D
                  1669    ;
                  1670    ;   Here we should have a number between 0-9
                  1671    ;
05A6 D0E0         1672    INH1A:  POP     ACC
05A8 C3           1673            CLR     C
05A9 9430         1674            SUBB    A,#30H
                  1675    ;   Check to see if we have a valid number between 0-9.
05AB C0E0         1676            PUSH    ACC
05AD C3           1677            CLR     C
05AE 940A         1678            SUBB    A,#0AH                  ; Should set carry
05B0 4002         1679            JC      INH1E                   ; Jump if good
05B2 D202         1680            SETB    INHF
05B4 D0E0         1681    INH1E:  POP     ACC
05B6 300202       1682    INH1D:  JNB     INHF,INH1G              ; Jump if no error
05B9 E55E         1683            MOV     A,STORE                 ; Restore ascii value if error
05BB 22           1684    INH1G:  RET
                  1685    ;
                  1686    ;
                  1687    ;-----------------------------------------------------------------------
                  1688    ;
                  1689    ;
                  1690    ;               ***************** OVFC ***************
                  1691    ;
                  1692    ;
                  1693    ;       This routine checks to see if the external buffer has overflowed.
```

```
LOC  OBJ            LINE    SOURCE

                    1694    ;       Overflow of the external buffer occurs when the put pointer equals
                    1695    ;       the get pointer (DPTR=R7,R6) after a byte has been put into the
                    1696    ;       external buffer by PUTIN.  If overflow occurs, this routine sets
                    1697    ;       the OVFF flag and then backs the put pointer (DPTR) up by 7.
                    1698    ;       This makes way so that an end of word and an end of file marker
                    1699    ;       can be put into the external buffer.  This routine does not put these
                    1700    ;       markers into the external buffer because this routine is called right
                    1701    ;       after PUTIN, and putting in these markers involves using PUTIN.
                    1702    ;       If the act of putting in one of these markers caused the overflow
                    1703    ;       and if this routine also put in these markers, we would get a
                    1704    ;       never ending cycle.  I don't want this to happen.
                    1705    ;
                    1706    ;
05BC C0E0           1707    OVFC:   PUSH    ACC
05BE EF             1708            MOV     A,R7
05BF B58345         1709            CJNE    A,DPH,OVF1
05C2 EE             1710            MOV     A,R6
05C3 B58241         1711            CJNE    A,DPL,OVF1
                    1712    ;
                    1713    ;   Here we have found that DPTR=R7,R6
                    1714    ;******************** EXTERNAL RAM OVERFLOW ********************
                    1715    ;
05C6 D20F           1716            SETB    OVFF                    ; Indicate overflow
                    1717    ;   Must check if DPTR=XMBEG (external memory beginning)
05C8 E583           1718            MOV     A,DPH
05CA B4200A         1719            CJNE    A,#(HIGH XMBEG),$+2
05CD E582           1720            MOV     A,DPL
05CF B40005         1721            CJNE    A,#(LOW XMBEG),OVF2
                    1722    ;   Here we have found that DPTR=XMBEG
05D2 903FF9         1723            MOV     DPTR,#(XMEND-7)
05D5 C107           1724            AJMP    OVF1
                    1725    ;
                    1726    ;   Here we have found that DPTR<>XMBEG
                    1727    ;
05D7 E582           1728    OVF2:   MOV     A,DPL
05D9 C3             1729            CLR     C
05DA 9407           1730            SUBB    A,#7
05DC F582           1731            MOV     DPL,A
05DE E583           1732            MOV     A,DPH
05E0 9400           1733            SUBB    A,#0                    ; Subtract any carry
05E2 F583           1734            MOV     DPH,A
                    1735    ;
                    1736    ;   Now we must check to see if DPTR is within ext buffer range after the
                    1737    ;   subtraction.
                    1738    ;
05E4 7400           1739            MOV     A,#(LOW XMBEG)
05E6 C3             1740            CLR     C
05E7 9582           1741            SUBB    A,DPL
05E9 F55E           1742            MOV     STORE,A                 ; Store possible LO byte to subtract
                    1743                                            ; from XMEND.
```

```
LOC  OBJ            LINE     SOURCE

05EB 7420           1744             MOV     A,#(HIGH XMBEG)
05ED 9583           1745             SUBB    A,DPH
05EF 4016           1746             JC      OVF1                    ; Jump if DPTR > XMBEG (all's ok)
                    1747     ;
                    1748     ;  Here DPTR is not within ext buffer range after subtracting 7 above.
                    1749     ;
05F1 904000         1750             MOV     DPTR,#XMEND
05F4 C0E0           1751             PUSH    ACC                     ; Store HO byte to subtract from
                    1752                                             ; XMEND.
05F6 E582           1753             MOV     A,DPL
05F8 C3             1754             CLR     C
05F9 955E           1755             SUBB    A,STORE
05FB F582           1756             MOV     DPL,A
05FD D0E0           1757             POP     ACC                     ; Get HO byte to subtract from XMEND.
05FF F55E           1758             MOV     STORE,A
0601 E583           1759             MOV     A,DPH
0603 955E           1760             SUBB    A,STORE
0605 F583           1761             MOV     DPH,A
                    1762     ;
                    1763     ;  The data pointer now has the correct memory value, within the ext buffer
                    1764     ;  range.
                    1765     ;
0607 D0E0           1766     OVF1:   POP     ACC
0609 22             1767             RET
                    1768     ;
                    1769     ;
                    1770     ;
                    1771     ;-------------------------------------------------------------------------
                    1772     ;
                    1773     ;VERSION 4.21
                    1774     ;JIM KEPLER
                    1775     ;5/30/85
                    1776     ;
                    1777     ;-------------------------------------------------------------------------
                    1778
                    1779             END
```

SYMBOL TABLE LISTING

NAME	TYPE	VALUE	ATTRIBUTES
ACC. . . .	D ADDR	00E0H	A
ACMD . . .	NUMB	002EH	A
ANAF . . .	B ADDR	0021H.6	A
ATH1 . . .	C ADDR	03D1H	A
ATH2 . . .	C ADDR	03CAH	A
ATHR . . .	D ADDR	005BH	A
ATHRC. . .	D ADDR	005CH	A
ATHRR. . .	C ADDR	03BBH	A
BCMD . . .	NUMB	0026H	A
BELOWT . .	C ADDR	03F2H	A
BF	B ADDR	0021H.0	A
BGN. . . .	C ADDR	01CEH	A
BOOT . . .	C ADDR	01E6H	A
BPF. . . .	NUMB	0007H	A
BPFZC. . .	NUMB	000FH	A
BTHRC. . .	D ADDR	005DH	A
BUFR . . .	B ADDR	0020H.1	A
BWORD. . .	C ADDR	0395H	A
CHECK. . .	C ADDR	03DFH	A
CNT1 . . .	C ADDR	02FEH	A
CNTR . . .	C ADDR	02EEH	A
CON1 . . .	C ADDR	0481H	A
CONVERT. .	C ADDR	046FH	A
CR	NUMB	000DH	A
DACMD. . .	NUMB	002CH	A
DATAOUT. .	C ADDR	0468H	A
DET1 . . .	C ADDR	035FH	A
DET2 . . .	C ADDR	037EH	A
DET5 . . .	C ADDR	03AAH	A
DET6 . . .	C ADDR	03B1H	A
DETECT . .	C ADDR	0353H	A
DP1. . . .	C ADDR	0544H	A
DPAR . . .	C ADDR	053DH	A
DPH. . . .	D ADDR	0083H	A
DPL. . . .	D ADDR	0082H	A
DSCMD. . .	NUMB	0029H	A
DSPCMD . .	NUMB	0028H	A
DSPF . . .	B ADDR	0021H.3	A
EOF. . . .	NUMB	0024H	A
EOWM . . .	NUMB	002AH	A
EWORD. . .	C ADDR	039FH	A
FIL1 . . .	C ADDR	0456H	A
FILM . . .	C ADDR	0444H	A
GDCMD. . .	NUMB	0022H	A
GDF. . . .	B ADDR	0021H.4	A
GET1 . . .	C ADDR	04C7H	A
GET2 . . .	C ADDR	04D5H	A

NAME	TYPE	VALUE	ATTRIBUTES
GETOUT . .	C ADDR	04A2H	A
GPAF	B ADDR	0022H.0	A
GTSCMD . .	NUMB	0025H	A
GTSF	B ADDR	0021H.2	A
HCRDY. . .	B ADDR	0020H.5	A
HEX. . . .	C ADDR	0484H	A
I1	C ADDR	0109H	A
I10. . . .	C ADDR	016EH	A
I10A . . .	C ADDR	017CH	A
I11. . . .	C ADDR	017EH	A
I11A . . .	C ADDR	0189H	A
I12. . . .	C ADDR	018BH	A
I12A . . .	C ADDR	0196H	A
I13. . . .	C ADDR	0198H	A
I13A . . .	C ADDR	01A3H	A
I14. . . .	C ADDR	01A5H	A
I14A . . .	C ADDR	01B0H	A
I15. . . .	C ADDR	01B2H	A
I1A. . . .	C ADDR	0113H	A
I2	C ADDR	011EH	A
I20. . . .	C ADDR	01C3H	A
I21. . . .	C ADDR	01B6H	A
I3	C ADDR	0127H	A
I3A. . . .	C ADDR	012FH	A
I4	C ADDR	0133H	A
I5	C ADDR	0143H	A
I6	C ADDR	014AH	A
I6A. . . .	C ADDR	0157H	A
I7	C ADDR	0159H	A
I8	C ADDR	0160H	A
I9	C ADDR	0167H	A
IE	D ADDR	00A8H	A
INH. . . .	C ADDR	0561H	A
INH1 . . .	C ADDR	057CH	A
INH1A. . .	C ADDR	05A6H	A
INH1B. . .	C ADDR	0596H	A
INH1C. . .	C ADDR	05A1H	A
INH1D. . .	C ADDR	05B6H	A
INH1E. . .	C ADDR	05B4H	A
INH1G. . .	C ADDR	05BBH	A
INHF . . .	B ADDR	0020H.2	A
INIP . . .	C ADDR	033DH	A
INIZC. . .	C ADDR	0327H	A
INPA . . .	C ADDR	051AH	A
INTR . . .	C ADDR	0100H	A
IP	B ADDR	00B8H	A
ITB1 . . .	C ADDR	034AH	A
ITBCLD . .	C ADDR	0346H	A
IWORD. . .	C ADDR	028BH	A

NAME	TYPE	VALUE	ATTRIBUTES
LOOP0. . .	C ADDR	023CH	A
LOOP1. . .	C ADDR	0269H	A
M1	C ADDR	0467H	A
MAYBE. . .	C ADDR	0457H	A
MUXST. . .	NUMB	0000H	A
NL	NUMB	000AH	A
NSTART . .	C ADDR	03FFH	A
NTSCMD . .	NUMB	003BH	A
NTSS . . .	D ADDR	005AH	A
NWORD. . .	C ADDR	039CH	A
OVF1 . . .	C ADDR	0607H	A
OVF2 . . .	C ADDR	05D7H	A
OVFC . . .	C ADDR	05BCH	A
OVFF . . .	B ADDR	0021H.7	A
OVFM . . .	NUMB	0025H	A
P1	D ADDR	0090H	A
P3	D ADDR	00B0H	A
PA1. . . .	C ADDR	0502H	A
PA2. . . .	C ADDR	0504H	A
PA3. . . .	C ADDR	0511H	A
PARAM. . .	C ADDR	04D8H	A
PCMD . . .	NUMB	0023H	A
PSW. . . .	D ADDR	00D0H	A
PUT1 . . .	C ADDR	04A2H	A
PUT2 . . .	C ADDR	04A8H	A
PUTIN. . .	C ADDR	048DH	A
REC. . . .	C ADDR	052DH	A
RI	B ADDR	0098H.0	A
RIF. . . .	B ADDR	0020H.4	A
RQCMD. . .	NUMB	000AH	A
RSO. . . .	B ADDR	00D0H.3	A
SBUF . . .	D ADDR	0099H	A
SCON . . .	D ADDR	0098H	A
SDCMD. . .	NUMB	0027H	A
SEN1 . . .	C ADDR	040EH	A
SEN2 . . .	C ADDR	042CH	A
SEN3 . . .	C ADDR	041AH	A
SEND . . .	C ADDR	0404H	A
SK1. . . .	C ADDR	013FH	A
SKIP0. . .	C ADDR	0255H	A
SMLP . . .	C ADDR	0290H	A
SMLP1. . .	C ADDR	0295H	A
SMLP2. . .	C ADDR	029AH	A
SMLP3. . .	C ADDR	029FH	A
SMLP4. . .	C ADDR	02A3H	A
SMLP5. . .	C ADDR	02D9H	A
SMLP6. . .	C ADDR	02DEH	A
SP	D ADDR	0081H	A
SPACE. . .	NUMB	0020H	A
SPCC . . .	D ADDR	0059H	A

```
NAME    TYPE  VALUE     ATTRIBUTES

SPER . . .  D ADDR    004EH   A
ST . . . .  B ADDR    0020H.7 A
STACK. . .  D ADDR    005FH   A
STCMD. . .    NUMB    0021H   A
STEN . . .  B ADDR    0020H.0 A
STLTM. . .  D ADDR    0050H   A
STORE. . .  D ADDR    005EH   A
STZC . . .  C ADDR    0314H   A
TBNEW. . .  D ADDR    0030H   A
TBOLD. . .  D ADDR    003FH   A
TCON . . .  D ADDR    0088H   A
TF0. . . .  B ADDR    0088H.5 A
TH0. . . .  D ADDR    008CH   A
TH1. . . .  D ADDR    008DH   A
THR1 . . .  D ADDR    0051H   A
THR2 . . .  D ADDR    0052H   A
THR3 . . .  D ADDR    0053H   A
THR4 . . .  D ADDR    0054H   A
THR5 . . .  D ADDR    0055H   A
THR6 . . .  D ADDR    0056H   A
THR7 . . .  D ADDR    0057H   A
TI . . . .  B ADDR    0098H.1 A
TIF. . . .  B ADDR    0020H.3 A
TL0. . . .  D ADDR    008AH   A
TL1. . . .  D ADDR    008BH   A
TMOD . . .  D ADDR    0089H   A
TR0. . . .  B ADDR    0088H.4 A
TRANS. . .  C ADDR    0535H   A
TSAF . . .  B ADDR    0021H.5 A
TSAM . . .  D ADDR    0059H   A
TSCMD. . .    NUMB    003AH   A
WBCMD. . .    NUMB    003DH   A
WBF. . . .  B ADDR    0021H.1 A
WBOOT. . .  C ADDR    0211H   A
WOR1 . . .  C ADDR    0443H   A
WORM . . .  C ADDR    0431H   A
XMBEG. . .    NUMB    2000H   A
XMEND. . .    NUMB    4000H   A
XMT. . . .  B ADDR    0020H.6 A

REGISTER BANK(S) USED: 0, TARGET MACHINE(S): 8051

ASSEMBLY COMPLETE, NO ERRORS FOUND
```

APPENDIX E

Z-151 TRAINJFK PROGRAM LISTING

```
{ ********************************************************************** }
{ **********                                          ********** }
{ **********  Voice Recognition System                ********** }
{ **********                                          ********** }
{ **********            TRAINJFK                      ********** }
{ **********                                          ********** }
{ ********************************************************************** }

PROGRAM TRAIN ( INPUT, OUTPUT );

{ Date: September 13,1984 }

{ Updated: May 31, 1985 - Revision 3.20 }

{ Author: Jim Kepler }

{ Coauthor: Trang Nguyen }

{ This is the program that runs on the host computer during }
{ the training phase of the voice recognition system.      }

{ ---------------------------------------------------------------------- }

LABEL   OUT;                    { Jump to OUT if 8751 ext buffer overflows }

CONST   END_CMD = '*';          { Command to signify user wants to end input }
        END_WORD = '*';         { End of word marker from F-E box }
        FE_START = '!';         { Start looking for word command to F-E box }
        MAX_SAMPLE = 65;        { Maximum number of frames per word }
        OVFM = '%';             { Overflow marker from F-E box }
        END_FILE = '$';         { End of file marker from F-E box }
        CP = '?';               { Command to signify user wants to change }
                                { the 8751 parameters }
        RINTEL = '=';           { Command to warm boot or reset the 8751 }
        CINTEL = '%';           { Command to cold boot the 8751 }
        WORM = '&';             { Command to signify user wants to boot 8751 }


TYPE    LINE_TYPE = ARRAY [1..40] OF CHAR;
                                { Input line buffer }
        PARA_TYPE = ARRAY [1..40] OF CHAR;
                                { Input line buffer, for parameter procedure }

VAR     VOCAB : TEXT;           { Vocabulary file  variable }
        VOCAB_FILE: STRING [19];{ Name of the vocab file on disk }
        WORD : STRING [50];     { Input word or end command }
        SAMPLE : ARRAY [1..MAX_SAMPLE] OF LINE_TYPE;
                                { Storage for the frames of a word }
        DUMMY : LINE_TYPE;      { Used when word has too many frames }
        TOO_LARGE : BOOLEAN;    { Indicates word has too many frames }
        N : INTEGER;            { Number of frames in this word }
        EW : INTEGER;           { Number of frames up to end of word marker }
        I, J : INTEGER;         { Loop count }
```

```
    ACTIVE : BOOLEAN;          { Indicates activity from F-E box }
    CH : CHAR;                 { Answer to yes/no question }
    MIN_SAMPLE : INTEGER;      { Minimum number of frames per word }
    SPC : INTEGER;             { Number of frames to chop off end of word }
    OVFF : BOOLEAN;            { 8751 ext buffer overflow flag }
    GOLF : BOOLEAN;            { Set when want to restart main loop }
    ANS : CHAR;                { Answer for choice of either to }
                               { warm boot or cold boot the 8751 }
    NL : CHAR;                 { Declare line feed character }

{ ------------------------------------------------------------------ }


PROCEDURE INITIALIZE;

  { Initialization procedure }
  { Set up vocabulary file on disk to store the frames of data }

  BEGIN
    NL := CHAR(10);            { Line feed character }
    WRITE('What is the name of the new vocabulary file ? : ');
    READLN(VOCAB_FILE);        { Name of the vocabulary file }
    ASSIGN(VOCAB,VOCAB_FILE);  { Associate VOCAB with vocabulary file }
    REWRITE(VOCAB);            { Initialize the vocabulary file }
    WRITE(OUTPUT,'Enter minimum frame count: ');
    READLN(INPUT,MIN_SAMPLE);  { Read min frame count }
    WRITE(OUTPUT,'Number of frames to chop off the end of word: ');
    READLN(INPUT,SPC)          { Read num of frames to chop off }
  END;

{ ------------------------------------------------------------------ }

PROCEDURE A872(VAR LINE:LINE_TYPE):EXTERNAL 'A872.COM';

  { This is an assembly language routine that will send out a request }
  { command (NL) to the F-E box and then it will collect a frame of   }
  { data, including the NL at the end of the frame.  When reading a   }
  { frame, this routine reads in bytes from the serial port, until a  }
  { NL character is received, in which case the procedure ends.       }

{ ------------------------------------------------------------------ }

PROCEDURE READLINE ( VAR LINE : LINE_TYPE );

  { Reads one frame into the variable LINE, this includes the NL }

  VAR   I : INTEGER;           { Index into LINE }

  BEGIN
    I := 1; OVFF := FALSE;     { Reset overflow flag }
    A872(LINE);                { Read in a frame of data }
    WHILE LINE[I] <> CHAR(10) DO  { While not NL }
      BEGIN
```

```
      IF LINE[I] = OVFM        { Check for overflow marker }
        THEN OVFF := TRUE;     { If overflow occurs then set the }
      I := I + 1;              { overflow flag. }
    END;
  END;

{ ------------------------------------------------------------------ }


PROCEDURE A87P(VAR TMPAR:PARA_TYPE); EXTERNAL 'A87P.COM';

  { This assembly language routine will read in the parameters of the }
  { 8751.  This is used in procedure READPARA.                        }


{ ------------------------------------------------------------------ }

PROCEDURE READPARA ( VAR LINEOUT:PARA_TYPE);

  { This procedure will read the strins of parameters from the 8751      }
  { into TMPAR.  The strins is terminated with a CR.  Then this procedure }
  { deletes all the blanks from the strins and puts the result in        }
  { LINEOUT.  Blanks occur after every two characters.                   }

CONST    BLANK = ' ';           { Blank character }

  VAR    C,D : INTEGER;
         TMPAR : PARA_TYPE;     { A temporary array that holds the }
                                { parameter data before any blanks }
                                { are deleted }

  BEGIN
    WRITELN(OUTPUT,'Now we are readins in the parameters');
    C := 1; D := 1;
    A87P(TMPAR);                { Read in whole line of parameters }
    WHILE TMPAR[C] <> CHAR(13) DO  { If not a CR }
      BEGIN                     { Delete all inserted blanks }
        IF TMPAR[C] <> BLANK THEN
          BEGIN
            LINEOUT[D] := TMPAR[C];
            D := D + 1;
          END;
        C := C + 1;
      END;
    LINEOUT[D] := CHAR(13);     { Insert a CR at the end }
  END;

{ ------------------------------------------------------------------ }


PROCEDURE A870(VAR TEMPINPUT:PARA_TYPE);EXTERNAL 'A870.COM';

  { This assembly language routine sends the first character in TEMPINPUT }
```

```
  { out the serial port to the 8751.  Then it waits for two bytes to be   }
  { sent, the first byte sent is put into TEMPINPUT[2], the second byte   }
  { sent is ignored.                                                      }

{ ------------------------------------------------------------------------ }

PROCEDURE PARAMETERS;

  { This procedure displays the current 8751 parameters, and then allows the }
  { user to input new values for these parameters.  Note the parameters      }
  { should be entered in hexadecimal format.  This procedure also checks     }
  { for serial communication errors.                                         }

  CONST   ACMD  = ',';          { Command for averaging adjacent frames }
          DACMD = '.';          { Command for not averaging adjacent frames }
          TSCMD = ':';          { Command for collecting 5 extra frames }
          NTSCMD = ';';         { Command for not collecting 5 extra frames }

  VAR     TEMP : ARRAY [1..7] OF STRING [80]; { Array to hold output strings }
          STOREP : PARA_TYPE;       { 8751 parameters stored here }
          DUMPARA : PARA_TYPE;
          TEMPINPUT : PARA_TYPE;
          PARVALUE : STRING [80]; { New parameter from user }
          VALID  : BOOLEAN;     { Flags if input parameter is valid }
          I, J   : INTEGER;
          P, KT  : INTEGER;
          NF     : INTEGER;
          ERRF   : BOOLEAN;     { serial communications error flag }
          COM    : CHAR;
  BEGIN
      { First initialize output string variables }
      TEMP[1]:= '62.5 Hz';TEMP[2]:= '125 Hz';TEMP[3]:= '250 Hz';
      TEMP[4]:= '500 Hz';TEMP[5]:= '1000 Hz';TEMP[6]:= '2000 Hz';
      TEMP[7]:= '4000 Hz';
      WRITELN(OUTPUT,'*********************************************');
      WRITELN(OUTPUT,'    You have made an excellent choice !!');
      WRITELN(OUTPUT,'You are entering the mighty parameter routine');
      WRITELN(OUTPUT,'*********************************************');
      READPARA(STOREP);         { Read in the line of parameters from 8751 }
      WRITELN(OUTPUT);
      WRITELN(OUTPUT, 'Current 8751 parameters ');
      WRITELN(OUTPUT);
      WRITE(OUTPUT,'Sampling period value (SPER) = ');
      FOR I := 1 TO 4           { Display the 4 bytes of SPER }
        DO WRITE(OUTPUT,STOREP[I]);
      WRITELN(OUTPUT);
      WRITE(OUTPUT,'Settling time constant for the log amp (STLTM) = ');
      FOR I := 5 TO 6           { Display the 2 bytes of STLTM }
        DO WRITE(OUTPUT,STOREP[I]);
      WRITELN(OUTPUT);
      NF := 7;                  { Pointer to thresholds within parameter line }
      FOR I := 1 TO 7 DO
```

```
  BEGIN                    { This loop will display the 7 PD thresholds }
    WRITE(OUTPUT,'THR',I,' = ',' Threshold value for band ',
          I,' (',TEMP[I],')   = ');
    FOR J := NF TO NF+1
      DO WRITE(OUTPUT,STOREP[J]);
    WRITELN(OUTPUT);
    NF := NF + 2           { Point to beginning of next threshold }
  END;
WRITE(OUTPUT,'SPCC = ');
FOR I := 21 TO 22          { This will display the 2 bytes of SPCC }
  DO WRITE(OUTPUT,STOREP[I]);
WRITELN(OUTPUT);
WRITE(OUTPUT,'TSAM = ');
FOR I := 23 TO 24          { This will display the 2 bytes of TSAM }
  DO WRITE(OUTPUT,STOREP[I]);
WRITELN(OUTPUT);
WRITELN(OUTPUT);
WRITELN(OUTPUT,'Input new parameters in hexadecimal format ');
WRITELN(OUTPUT);
WRITELN(OUTPUT);
VALID := FALSE;
WHILE (VALID = FALSE) DO
  BEGIN                    { Here we input a new value for SPER }
    WRITE(OUTPUT,'Sampling period value (SPER) =  ');
    READLN(INPUT,PARVALUE);  { Input new value }
    IF LENGTH(PARVALUE) > 5 THEN   { If too long }
      BEGIN
        VALID := FALSE;
        WRITELN(OUTPUT,'Exceeded the maximum length for SPER, Please retype! ');
      END
    ELSE VALID := TRUE;
  END;
IF COPY(PARVALUE,1,1) <> NL THEN  { If first char not NL }
  FOR I := 1 TO 4                 { Store new SPER in parameter line }
    DO STOREP[I] := COPY(PARVALUE,I,1);
VALID := FALSE;
WHILE (VALID = FALSE) DO
  BEGIN                    { Here we put in new value for STLTM }
    WRITE(OUTPUT,'Settling time constant for log amplifier (STLTM) =  ');
    READLN(INPUT,PARVALUE);       { Input new value }
    IF LENGTH(PARVALUE) > 3 THEN  { If too long }
      BEGIN
        VALID := FALSE;
        WRITELN(OUTPUT,'Exceeded the maximum length of STLTM, Please retype! ');
      END
    ELSE VALID := TRUE;
  END;
IF COPY(PARVALUE,1,1) <> NL THEN  { If first char not NL }
  BEGIN
    P := 1;
    FOR I := 5 TO 6
      DO BEGIN             { Store new STLTM in parameter line }
        STOREP[I] := COPY(PARVALUE,P,1);
```

```
        P := P + 1;
      END;
  END;
KT := 7;                    { Index to first threshold in parameter line }
  FOR I := 1 TO 7 DO
    BEGIN                   { This loop will read in the 7 new thresholds }
      VALID := FALSE;       { and make sure they are correct length       }
      WHILE (VALID = FALSE) DO      { Here, read in each threshold }
        BEGIN
          WRITE(OUTPUT,'TH',I,' = ','Threshold value for band ',
                I,' (',TEMP[I],') = ');
          READLN(INPUT,PARVALUE);           { Input new value }
          IF LENGTH(PARVALUE) > 3 THEN      { If too long }
            BEGIN
              VALID := FALSE;
              WRITE(OUTPUT,'Exceeded the maximum length (2 bytes) for ');
              WRITELN(OUTPUT,'the THRESHOLD value, Please retype! ');
            END
          ELSE VALID := TRUE;
        END;
        IF COPY(PARVALUE,1,1) <> NL THEN  { If input isnt NL }
           BEGIN
             P := 1;
             FOR J := KT TO KT+1 DO
               BEGIN        { Store new threshold in parameter line }
                 STOREP[J] := COPY(PARVALUE,P,1);
                 P := P + 1;
               END;
           END;
           KT := KT + 2     { Index to next threshold }
    END;
   VALID := FALSE;
    WHILE (VALID = FALSE) DO
      BEGIN                 { Input new SPCC value }
        WRITE(OUTPUT,'SPCC = ');
        READLN(INPUT,PARVALUE);             { Read new value }
        IF LENGTH(PARVALUE) > 3 THEN        { If too long }
          BEGIN
            VALID := FALSE;
            WRITE(OUTPUT,'Exceed the maximum length of (2 bytes) ');
            WRITELN(OUTPUT,'for SPCC, Please retype! ');
          END
        ELSE VALID := TRUE;
      END;
    IF COPY(PARVALUE,1,1) <> NL THEN        { If first char not NL }
      BEGIN
      P := 1;
      FOR I := 21 TO 22
        DO BEGIN            { Store new SPCC in parameter line }
          STOREP[I] := COPY(PARVALUE,P,1);
          P := P+1
        END;
      END;
```

```
      VALID := FALSE;
       WHILE (VALID = FALSE) DO
         BEGIN              { Here we input new TSAM value }
          WRITE(OUTPUT,'TSAM = ');
          READLN(INPUT,PARVALUE);        { Input new value }
          IF LENGTH(PARVALUE) > 3 THEN   { If too long }
           BEGIN
            VALID := FALSE;
            WRITELN(OUTPUT,'Exceeded the maximum length of (2 bytes) for',
                    ' the TSAM value, Please retype! ');
           END
          ELSE VALID := TRUE;
         END;
       IF COPY(PARVALUE,1,1) <> NL THEN     { If first char not NL }
        BEGIN
          P := 1;
          FOR I := 23 TO 24
           DO BEGIN        { Store new TSAM value in parameter line }
            STOREP[I] := COPY(PARVALUE,P,1);
            P := P + 1;
           END;
        END;

{ Send out the new parameters to 8751 and compare for }
{ communication for error. }

     ERRF := FALSE;
     FOR I := 1 TO 24
       DO BEGIN
        TEMPINPUT[1]:=STOREP[I];
        TEMPINPUT[2]:='@';
        A870(TEMPINPUT);    { Send out first byte and receive echo }
        IF TEMPINPUT[1] <> TEMPINPUT[2] THEN
          BEGIN
           WRITELN(OUTPUT,'**************************************');
           WRITELN(OUTPUT,'I am sorry to inform you that we');
           WRITELN(OUTPUT,'have a serial communication error!');
           WRITELN(OUTPUT,'**************************************');
           ERRF := TRUE;
          END;
        IF ERRF = TRUE THEN I := 24;
       END;

{ This part of the parameter procedure will give the user some options,    }
{ and then will issue the commands to the 8751 based on user response.     }

      IF  ERRF = FALSE THEN   { If no communication error }
       BEGIN
        WRITELN(OUTPUT,'Average adjacent frames (Y/N) ? ');
        READLN(INPUT,COM);
        IF (COM = 'Y') OR (COM = 'y') THEN
          WRITE(LST,ACMD);
        IF (COM = 'N') OR (COM = 'n') THEN
```

```
          WRITE(LST,DACMD);
        WRITELN(OUTPUT,'Collect 5 frames after the end of word (Y/N) ? ');
        READLN(INPUT,COM);
        IF (COM = 'Y') OR (COM = 'y') THEN
          WRITE(LST,TSCMD);
        IF (COM = 'N') OR (COM = 'n') THEN
          WRITE(LST,NTSCMD);
      END;
  END;
```

```
{ ------------------------------------------------------------------------ }

  { This is the main program, responsible for running the training phase     }
  { of the voice recognition system.  From here, the user can give a         }
  { command to change the 8751s parameters, or the user can boot the 8751.   }
  { This program also checks to see if the frame count is between MAX_SAMPLE }
  { and MIN_SAMPLE.  If it isn't, appropriate actions are taken; if it is,   }
  { the frames are stored in the vocabulary file.                            }

BEGIN  { Main program }
  WRITELN(OUTPUT,'*********************************************************');
  WRITELN(OUTPUT,'********** The training phase has started **********');
  WRITELN(OUTPUT,'*********************************************************');
  INITIALIZE;                   { Initialize this program }
  REPEAT                        { Command loop }
    GOLF := FALSE;
    WRITELN(OUTPUT,'Type in the word ("*" to exit or "?" to');
    WRITE(OUTPUT,'change 8751 parameters or "&" to boot 8751): ');
    READLN(INPUT,WORD);         { Get typed input }
    IF WORD = CP THEN           { If we want to change the perameters }
       PARAMETERS               { Change 8751 parameters }
    ELSE IF WORD = WORM THEN    { Here the user wants to boot the 8751 }
       BEGIN
         WRITE(OUTPUT,'Warm boot the 8751  (Y/N) ? ');
         READLN(INPUT,ANS);
         IF (ANS = 'Y') OR (ANS = 'y') THEN
           WRITE(LST,RINTEL)
         ELSE IF (ANS = 'N') OR (ANS = 'n') THEN
           BEGIN
             WRITE(OUTPUT,'Cold boot the 8751  (Y/N) ? ');
             READLN(INPUT,ANS);
             IF (ANS = 'Y') OR (ANS = 'y') THEN
               WRITE(LST,CINTEL);
           END;
       END
    ELSE IF WORD <> END_CMD THEN  { If we didn't type the end command }
      BEGIN
```

```
N := 0;                 { Initialize frame count }
WRITELN(OUTPUT,'Say the word typed above into the microphone.');
WRITELN(OUTPUT,'Hey bud, this means YOU ! ');
WRITE(LST,FE_START);    { Tell F-E box to start looking for a word }
TOO_LARGE := FALSE;     { Init flags }
ACTIVE := FALSE;
OVFF:= FALSE;
DUMMY[1] := ' ';        { Init first byte of dummy frame }
REPEAT                  { Here we read in all the frames up to the }
                        { end of word marker.                      }
  IF N < MAX_SAMPLE THEN
    BEGIN               { This executes when not too many frames }
      N := N + 1;       { Update frame count }
      READLINE(SAMPLE[N]);
                        { Read one frame from the F-E box }
      IF OVFF = TRUE THEN   { If 8751 ext buffer overflow }
        BEGIN
          WRITELN(OUTPUT,'8751 ext buffer has overflowed !! ');
          WRITE(LST,RINTEL); { Reset the 8751 }
          GOLF := TRUE; { Flag to get out of this loop and }
                        { restart the main loop. }
          IF GOLF = TRUE THEN GOTO OUT;
        END;
      IF NOT ACTIVE THEN
        BEGIN           { Executes only when we get the first frame }
          WRITELN(OUTPUT,'***** Something detected at Mic *****');
          ACTIVE := TRUE
        END;
    END
  ELSE
    BEGIN               { Executes when we get too many frames }
      READLINE(DUMMY);  { Throw away frame since too many frames }
      IF OVFF = TRUE THEN
        BEGIN           { Executes only if 8751 ext buf has overflowed }
          WRITELN(OUTPUT, '8751 ext buffer overflow !! ');
          WRITE(LST,RINTEL);     { Reset 8751 }
          GOLF := TRUE;
          IF GOLF = TRUE THEN GOTO OUT;   { Restart main loop }
        END;
      TOO_LARGE := TRUE { Indicate too many frames }
    END;
UNTIL (SAMPLE[N,1]=END_WORD) OR { Until end of word marker found }
      (DUMMY[1]=END_WORD);
EW := N;                { EW holds frame count up to end of word }
                        { marker.                                }
IF NOT GOLF THEN
  BEGIN
   IF N < MIN_SAMPLE THEN
     BEGIN              { Here we detect we have too few frames }
       WRITELN(OUTPUT,'Word too short!  Input ignored.');
       WRITE(LST,RINTEL);
     END                { Send warm boot command to 8751 }
   ELSE IF TOO_LARGE THEN
```

```
  BEGIN              { Here we detect we have too many frames }
    WRITELN(OUTPUT,'Word too long!  Input ignored.');
    WRITE(LST,RINTEL); { Reset the 8751 }
  END
ELSE
  BEGIN
    REPEAT           { Here we read frames after the end of word }
                     { marker, up till the end of file marker    }
      N := N + 1;    { Keep track of the total number of frames  }
      READLINE(SAMPLEIN);   { Read a frame of data }
      IF OVFF = TRUE THEN    { If 8751 ext buffer overflow }
        BEGIN
          WRITELN(OUTPUT, '8751 ext buffer overflow !!');
          WRITE(LST,RINTEL); { Reset 8751 }
          GOLF := TRUE;
          IF GOLF = TRUE THEN GOTO OUT;
        END;
    UNTIL (SAMPLEIN,1] = END_FILE); { Until end of file }
    IF NOT GOLF THEN
      BEGIN
        WRITELN(OUTPUT,'Would you like to store this prototype? ');
        READLN(INPUT,CH);            { Get answer }
        IF (CH='Y') OR (CH='y')
          THEN
            BEGIN     { Here we store the frames into the }
                      { vocabulary file                   }
              WRITELN(OUTPUT,'Storing ',WORD);
              WRITELN(VOCAB,WORD);   { Store text word in file }
               IF (N-EW) > SPC THEN  { If the number of frames }
               { after end of word marker is greater than the  }
               { number of frames to chop off the word         }
                  BEGIN
                    FOR I := 1 TO N-SPC-1 DO
                      BEGIN
                        J := 0;
                         REPEAT
                           J := J + 1;
                           WRITE(VOCAB,SAMPLE[I,J]) { Each byte }
                         UNTIL SAMPLE[I,J]=NL; { Until end of frame }
                      END;
                    WRITE(VOCAB,END_FILE);
                    { Insert end of file marker in vocab file }
                    WRITE(VOCAB,NL);
                  END
                ELSE BEGIN  { If the number of frames after end }
                { of word marker is less than the number of     }
                { frames to chop off the word                   }
                  FOR I := 1 TO N-SPC-2 DO
                    BEGIN
                      J := 0;
                        REPEAT
                          J := J + 1;
                          WRITE(VOCAB,SAMPLE[I,J]) { Each byte }
```

```
                        UNTIL SAMPLE[I,J]=NL; { Until end of frame }
                      END;
                    WRITE(VOCAB,END_WORD);
                    { Insert end of word marker in vocab file }
                    WRITE(VOCAB,NL);
                    WRITE(VOCAB,END_FILE);
                    { Insert end of file marker in vocab file }
                    WRITE(VOCAB,NL);
                  END;
              END
            ELSE WRITELN(OUTPUT,'Input ignored!')
          END
        END
      END;
OUT: END                { Jump here when 8751 ext buffer overflows }
  UNTIL WORD = END_CMD;  { Until end command detected }
  WRITELN(OUTPUT,'The training phase has now ended.');
  CLOSE(VOCAB);
END. { 5-31-85 - JFK }
```

```
; PROCEDURE A872.ASM

CODE    SEGMENT
        ASSUME  CS:CODE
A87     PROC    NEAR
        PUSH    BP
        MOV     BP,SP
        LES     DI,[BP+4]
        MOV     DX,03FDH
        IN      AL,DX
        AND     AL,01
        JZ      OVER2
        MOV     DX,03F8H            ;READ PORT TO CLEAR ANY GARBAG
        IN      AL,DX
OVER2:  MOV     DX,03F8H            ;OUTPUT PORT ADDRESS
        MOV     AL,0AH
        OUT     DX,AL               ;SEND <NL> OUT TO PORT
        MOV     DX,03FDH            ;ADDRESS OF LINE STATUS
L2:     IN      AL,DX               ;GET STATUS
        AND     AL,20H              ;SEE IF WORD HAS BEEN SENT
        JZ      L2                  ;IF NOT WAIT TILL IT HAS BEEN
L3:     MOV     DX,03FDH
L1:     IN      AL,DX               ;GET STATUS
        AND     AL,01               ;LOOK AT ONLY DATA READY
        JZ      L1                  ;KEEP POLLING TILL CHAR RECEIVED
        MOV     DX,03F8H            ;
        IN      AL,DX               ;GET CHAR
        MOV     ES:BYTE PTR[DI],AL  ;STORE IN ARRAY
        INC     DI                  ;GO TO NEXT SPACE IN ARRAY
        CMP     AL,0AH              ;CHECK FOR NEW LINE
        JE      OUT1                ;IF SO, EXIT
OVER1:  CMP     AL,0DH              ;CHECK FOR <CR>
        JNE     L3                  ;IF SO, EXIT
OUT1:   MOV     SP,BP
        POP     BP
        RET     4
A87     ENDP
CODE    ENDS
        END
```

```
; PROCEDURE A87D.ASM

CODE    SEGMENT
        ASSUME  CS:CODE
A87     PROC    NEAR
        PUSH    BP
        MOV     BP,SP
        LES     DI,[BP+4]
        MOV     DX,03FDH
        IN      AL,DX
        AND     AL,01H
        JZ      OVER2
        MOV     DX,03F8H
        IN      AL,DX
OVER2:  MOV     AL,ES:BYTE PTR[DI]      ;GET FIRST CHAR
        MOV     DX,03F8H                ;OUTPUT PORT ADDRESS
        OUT     DX,AL                   ;OUTPUT CHAR TO PORT
        MOV     DX,03FDH                ;ADDRESS OF LINE STATUS
L2:     IN      AL,DX                   ;GET STATUS
        AND     AL,20H                  ;SEE IF WORD HAS BEEN SENT
        JZ      L2                      ;IF NOT WAIT TILL IT HAS BEEN
        INC     DI                      ;ADDRESS OF NEXT BYTE IN ARRAY
L1:     MOV     DX,03FDH
        IN      AL,DX                   ;GET STATUS
        AND     AL,01H
        JZ      L1
        MOV     DX,03F8H
        IN      AL,DX
        MOV     ES:BYTE PTR[DI],AL
        MOV     DX,03FDH
L4:     IN      AL,DX
        AND     AL,01H
        JZ      L4
        MOV     DX,03F8H
        IN      AL,DX
        MOV     SP,BP
        POP     BP
        RET     4
A87     ENDP
CODE    ENDS
        END
```

```
; PROCEDURE A87P.ASM

CODE    SEGMENT
        ASSUME  CS:CODE
A87     PROC    NEAR
        PUSH    BP
        MOV     BP,SP
        LES     DI,[BP+4]
        MOV     DX,03FDH
        IN      AL,DX
        AND     AL,01
        JZ      OVER2
        MOV     DX,03F8H            ;READ BUFFER TO CLEAR ANY GARB
        IN      AL,DX
OVER2:  MOV     DX,03F8H            ;OUTPUT PORT ADDRESS
        MOV     AL,23H
        OUT     DX,AL               ;SEND '#' OUT TO PORT
        MOV     DX,03FDH            ;ADDRESS OF LINE STATUS
L2:     IN      AL,DX               ;GET STATUS
        AND     AL,20H              ;SEE IF WORD HAS BEEN SENT
        JZ      L2                  ;IF NOT WAIT TILL IT HAS BEEN
L3:     MOV     DX,03FDH
L1:     IN      AL,DX               ;GET STATUS
        AND     AL,01               ;LOOK AT ONLY DATA READY
        JZ      L1                  ;KEEP POLLING TILL CHAR RECEIVED
        MOV     DX,03F8H            ;
        IN      AL,DX               ;GET CHAR
        MOV     ES:BYTE PTR[DI],AL  ;STORE IN ARRAY
        INC     DI                  ;GO TO NEXT SPACE IN ARRAY
        CMP     AL,0AH              ;CHECK FOR NEW LINE
        JE      OUT1                ;IF SO, EXIT
OVER1:  CMP     AL,0DH              ;CHECK FOR <CR>
        JNE     L3                  ;IF SO, EXIT
OUT1:   MOV     SP,BP
        POP     BP
        RET     4
A87     ENDP
CODE    ENDS
        END
```

APPENDIX F

Z-151 RMATCH5 PROGRAM LISTING

```
{ ********************************************************************** }
{ ********************                              ******************** }
{ ********************  Voice Recognition System    ******************** }
{ ********************                              ******************** }
{ ********************          RMATCH5             ******************** }
{ ********************                              ******************** }
{ ********************          6-04-85             ******************** }
{ ********************************************************************** }

PROGRAM RECOGN ( INPUT, OUTPUT );

CONST   INF_DIST = 32767;       { LARGEST INTEGER VALUE }
        HSCORE   = 255;
        MAX_SAMPLE = 55;        { MAXIMUM SAMPLES PER WORD }
        N_BPF = 7;              { NUMBER OF ENERGY BANDS }
        N_ZC = 8;               { NUMBER OF ZERO CROSSING BANDS (INCLUDES MIC) }
        N_ELM = 15;             { N_BPF+N_ZC, NUMBER OF ELEMENTS IN A FRAME }
        MAX_VOCAB =36;          { MAXIMUM LIBRARY SIZE }
        FE_START = '!';         { STARTS F-E CONVERSION }
        END_WORD = '*';         { END OF WORD MARKER }
        END_FILE = '$';         { END OF FILE MARKER }

TYPE    TIME_SAMPLE = ARRAY [1..N_ELM] OF BYTE; { ONE FRAME OF DATA }
        SAMPLE = ARRAY [1..MAX_SAMPLE] OF TIME_SAMPLE; { WORD PATTERN }
        LINE_TYPE = ARRAY [1..40] OF CHAR;      { INPUT LINE BUFFER }
        BAND_SAMPLE = ARRAY [1..MAX_VOCAB,1..N_BPF] OF INTEGER;
        WS_SAMPLE   = ARRAY [1..MAX_VOCAB] OF INTEGER;

VAR     VOCAB : TEXT;           { LIBRARY FILE, HOLDS PATTERNS FOR WORDS }
        VOCAB_FILE : STRING[19]; {INPUT LIBRARY FILE NAME }
        FILENAME2 : STRING[19]; { COMPARISON FILE NAME }
        VOCAB2    : TEXT;       { COMPARISON FILE WORD PATTERNS }
        C_FILES   : ARRAY[1..10] OF STRING[12]; { DESIRED COMPARISON FILES }
        CMD1,CMD2,CMD3,CMD16,CMD15 : CHAR;      { USER COMMANDS }
        NL    : CHAR;           { REPRESENTS ASCII VALUE FOR A NEWLINE }
        OV_FLAG : INTEGER;      { 8751 BUFFER OVERFLOW FLAG }
        N_WORDS : INTEGER;      { NUMBER OF WORDS IN LIBRARY }
        WORDS : ARRAY [1..MAX_VOCAB] OF
          RECORD                {DATA STRUCTURE FOR HOLDING EACH WORD }
            N   : INTEGER;      { NUMBER OF FRAMES }
            TXT : STRING [20];  { TEXT STRING OF THIS WORD }
            DAT : SAMPLE        { WORD PATTERN }
          END;
        M_ADJ : INTEGER;        { MAXIMUM BOUNDARY ADJUSTMENT FOR TIME-WARPING }
        MIN_SAMPLE : INTEGER;   { MINIMUM NUMBER OF FRAMES PER WORD }
        SPC : INTEGER;          { SILENCE PERIOD COUNT FROM FE BOX }
        ZC_VAL : INTEGER;       { ZERO CROSSING TIME-WARPING SCORE }
        ZC_D   : ARRAY [1..MAX_SAMPLE,1..MAX_SAMPLE] OF INTEGER;
                                { ZERO CROSSING TIME-WARPING DISTANCE ARRAY }
        D      : ARRAY [1..MAX_SAMPLE,1..MAX_SAMPLE] OF INTEGER;
                                { BAND PASS ENERGY TIME-WARPING DISTANCE ARRAY }
        C_INDX   : INTEGER;     { INDEX IN C_FILES TO DENOTE WHAT COMPARISON }
                                { FILE IS CURRENTLY BEING USED }
```

```
        WIDX     : INTEGER;      { INDEX WHICH DENOTES WHAT WORD INTO THE }
                                 { COMPARISON FILE IS BEING LOOKED AT }
        BDSUM    : BAND_SAMPLE;  { HOLDS THE TOTAL ENERGY IN EACH BAND OF EACH WORD }
        WDSUM    : WS_SAMPLE;    { HOLDS THE TOTAL ENERGY OF EACH WORD }
        E_OR_ZC  : CHAR;         { FLAG FOR DENOTING WHETHER ENERGY OR ZERO }
                                 { CROSSINGS ARE SUPPOSED TO BE TIME-WARPED }
        WARPFR   : INTEGER;      { HOLDS WHAT BAND THE WARPING IS TO START AT }
        WARPFR2  : INTEGER;      { HOLDS WHAT BAND THE WARPING IS TO END AT }
        THR1     : INTEGER;      { UPPER THRESHOLD FOR MATCH1 SCORES }
        THR2     : INTEGER;      { UPPER THRESHOLD FOR MATCH2 SCORES }
        THR3     : INTEGER;      { UPPER THRESHOLD FOR MATCH3 SCORES }
        THR4     : INTEGER;      { UPPER THRESHOLD FOR MATCH4 SCORES }
        FRM_THR  : INTEGER;      { MAXIMUM ALLOWABLE FRAME DIFFERENCE BETWEEN WORDS }
        REC_DIF2 : INTEGER;      { DIFFERENCE THRESHOLD FOR GOING TO VERIFY STAGE }
        REC_DIF3 : INTEGER;      { DIFFERENCE THRESHOLD FOR DETERMING A WORD }
                                 { AFTER STAGE 3 }
        REC_DIF4 : INTEGER;      { DIFFERENCE THRESHOLD FOR DETERMING A WORD }
                                 { AFTER STAGE 4 }
        RIGHTON  : INTEGER;      { FLAG TO INDICATE IF LIBRARY SHOULD BE WRITTEN }
                                 { BACK TO THE DISK FILE }
        NUM      : INTEGER;      { NUMBER FOR WEIGHTED AVERAGING OF WAVES }
        MNSFLAG  : INTEGER;      { INDICATES IF AN ERRANT READING WAS READ }
                                 { FROM DISK FOR A WORD PATTERN }
        FULLFLAG : INTEGER;      { FLAGS FOR INDICATING IF THE LIBRARY }
        FULLFLAG2 : INTEGER;     { IS FULL }

{ ---------------------------------------------------------------- }

PROCEDURE READLINE ( VAR F : TEXT; VAR LINE : LINE_TYPE );

{ READS IN ONE LINE (FRAME) INTO AN ARRAY OF CHAR (LINE), TERMINATING
  WITH <NL> FROM THE TEXT FILE PASSED THROUGH F }

VAR I : INTEGER;

BEGIN
  I := 0;
  REPEAT
    I := I + 1;
    READ(F,LINE[I]);
  UNTIL LINE[I]=NL;
END;

{ ---------------------------------------------------------------- }

PROCEDURE CONVERT ( LINE : LINE_TYPE; VAR ARY : SAMPLE; S : INTEGER;
                    VAR MNSFLAG : INTEGER);

{ CONVERT THE INPUT ASCII CHARS IN LINE (REPRESENTS ONE FRAME OF DATA) FROM
  ASCII TO ITS NUMERIC VALUE AND STORES IN ROW S OF PASSED ARY.
  IF % IS FOUND THEN A BUFFER OVERFLOW CONDITION OCCURRED IN THE 3751 BUFFER
  AND IT FLAGS THIS BY SETTING THE OV_FLAG=1. IF AT ANY TIME A NEGATIVE
  VALUE IS READ FOR AN ENERGY OR ZERO CROSSING VALUE, OR A VALUE WHICH IS
```

```
  GREATLY DIFFERENT THAN IN THE PREVIOUS FRAME, THEN THE MNSFLAG IS SET.
  THIS HAPPENS WHEN A BYTE HAS BEEN LOST FROM A FRAME DURING THE TRAINING
  PHASE }

VAR     I     : INTEGER;
        ITNEG : INTEGER;

FUNCTION HEX ( CH : CHAR ) : INTEGER;

{ CONVERT HEX DIGIT IN ASCII TO INTEGER }

BEGIN
  IF (CH>='0') AND (CH<='9')
    THEN HEX := ORD(CH) - ORD('0')        { 0..9 }
    ELSE HEX := ORD(CH) - ORD('A') + 10;  { A..F }
  END;

BEGIN
  MNSFLAG:=0;
  FOR I := 1 TO N_ELM DO BEGIN
    IF (LINE[I+I-1]='%') OR (LINE[I+I]='%') THEN OV_FLAG:=1;
    ITNEG := HEX(LINE[I+I-1]) * 16 + HEX(LINE[I+I]);
    ARY[S,I]:=ITNEG;
    IF S > 1 THEN IF ABS(ARY[S-1,I]-ITNEG) > 50 THEN MNSFLAG:=1;
    IF ITNEG < 0 THEN MNSFLAG:=1;
  END;
END;

{ ---------------------------------------------------------------- }

PROCEDURE INITIALIZE;

{ READ IN LIBRARY FILE.  SET PARAMETERS TO THEIR DEFAULT VALUES.  }

  VAR I,J,K : INTEGER;          { LOOP VARIABLES }
      NAME1 : STRING[20];       { TEXT STRING OF A LIBRARY WORD }
      NAMED : STRING[20];       { THE TEXT STRING IN UPPERCASE }
      CHR   : CHAR;             { STORAGE VARIABLE }
      LINE : LINE_TYPE;         { AN ARRAY TO HOLD A FRAME OF DATA }

  BEGIN
    WRITE('WHAT IS THE LIBRARY FILE NAME ? ');
    READLN(VOCAB_FILE);
    ASSIGN(VOCAB,VOCAB_FILE);
    RESET(VOCAB);
    FOR K:=1 TO MAX_SAMPLE DO   { CLEAR ARRAYS D AND ZC_D }
      FOR J:=1 TO MAX_SAMPLE DO { TURBO DOES NOT CLEAR ARRAYS }
      BEGIN
        D[K,J]:=0;
        ZC_D[K,J]:=0;
      END;
    FOR K:=1 TO 40 DO           { CLEAR ARRAY LINE }
      LINE[K]:=' ';
```

```
  NL := CHAR(10);              { SET NL TO ASCII NEWLINE }
  RIGHTON :=0;
  NUM := 4;
  THR1:=40;
  THR2:=25;
  THR3:=8;
  REC_DIF2:=8;
  REC_DIF3:=5;
  REC_DIF4:=4;
  THR4:=10;
  FRM_THR:=10;
  M_ADJ := 1;
  MIN_SAMPLE := 1;
  N_WORDS := 0;
  SPC:=0;
  {WRITE(OUTPUT,'ENTER SILENT PERIOD COUNT: '); }
  {READLN(INPUT,SPC);                                }
  WRITELN(OUTPUT,'READING IN LIBRARY.  PLEASE WAIT...');
  REPEAT
    N_WORDS := N_WORDS + 1;
    READLN(VOCAB,NAME1);
    NAMED:='';
    FOR K:=1 TO LENGTH(NAME1) DO BEGIN
      CHR:=COPY(NAME1,K,1);
      CHR:=UPCASE(CHR);
      NAMED:=CONCAT(NAMED,CHR);
    END;
    WORDS[N_WORDS].TXT:=NAMED;
    WRITE(WORDS[N_WORDS].TXT:8); { WRITE WORD OUT TO SCREEN }
    I := 0;
    READLINE(VOCAB,LINE);
    WHILE LINE[1] <> END_FILE DO
      BEGIN
        IF LINE[1] <> END_WORD THEN
          BEGIN
            I := I + 1;
            CONVERT(LINE,WORDS[N_WORDS].DAT,I,MNSFLAG);
            IF MNSFLAG=1 THEN I:=I-1;
          END;
        READLINE(VOCAB,LINE); { Read in next frame }
      END;
    WORDS[N_WORDS].N := I
  UNTIL EOF(VOCAB) OR (N_WORDS = MAX_VOCAB);
  WRITELN;
{WRITE(AUX,'='); }              Clear 8751 buffers }
IF N_WORDS = MAX_VOCAB THEN BEGIN
  FULLFLAG2:=1;
  FULLFLAG := 1;
END
ELSE BEGIN
  FULLFLAG2:=0;
  FULLFLAG := 0;
END;
```

```
  WRITE('SHOULD INPUT WAVES BE AVERAGED ? ');
  READLN(CMD3);
  WRITE('SHOULD LIBRARY WAVE AVERAGING BE DONE ? ');
  READLN(CMD16);
  WRITE('SHOULD RESULTS GO TO LST DEVICE ? ');
  READLN(CMD15);
  WRITELN('WHAT MODE WOULD YOU LIKE THE LIBRARIAN IN ? ');
  WRITE('A - AUTOMATIC    0 - OFF ');
  READLN(CMD1);
END;

{ ---------------------------------------------------------------- }

PROCEDURE TOTAL_DIST ( ARY1 : SAMPLE; A : INTEGER;
                       ARY2 : SAMPLE; B : INTEGER;
                       VAR VAL:INTEGER; VAR ZVAL:INTEGER;
                       E_OR_ZC:CHAR);

{ SUBROUTINES WITHIN TOTAL_DIST ARE DIST, GET_D, DP_FCN, SET_BOUNDS
  VARIABLE PASSED ARE THE TWO WORD PATTERNS TO BE COMPARED, ARY1 AND ARY2,
  AND THE NUMBER OF FRAMES IN EACH PATTERN, A AND B RESPECTIVELY.
  A FLAG DENOTING WHETHER THE ENERGY OR ZERO CROSSING VALUES ARE TO BE
  COMPARED, E_OR_ZC, AND TWO VARIABLES IN WHICH TO RETURN THE
  CALCULATED TIME-WARP SCORE, VAL (FOR ENERGY) AND ZVAL (FOR ZERO-CROSSING).
  NOTE THAT GLOBAL VARIABLES WARPFR AND WARPFR2 SHOULD BE SET TO DENOTE WHAT
  BANDS SHOULD BE WARPED. TOTAL_DIST COMPUTES THE TIME-WARP SCORE BETWEEN ARY1
  AND ARY2, FOR THE DESIRED BANDS, AND RETURNS THIS SCORE IN VAL OR ZVAL
  (DEPENDING UPON WHETHER ENERGIES OR ZERO CROSSINGS WERE BEING WARPED).
  NOTE THAT A SHOULD BE GREATER THAN OR EQUAL TO B. }


VAR     X, Y : INTEGER;          { ARRAY INDICES }
        S : REAL;                { SLOPE OF BAND LIMETS }
        Y_MIN, Y_MAX : ARRAY [1..MAX_SAMPLE] OF INTEGER;
                                 { BOUNDS OF Y AS A FUNCTION OR X }

{---------------------------------------------------------------- }

PROCEDURE DIST (I1,I2,WARPFR,WARPFR2 : INTEGER; E_OR_ZC:CHAR);

{ SET D[I1,I2]= ABSOLUTE DIFFERENCE BETWEEN THE BAND WITH THE GREATEST
  DIFFERENCE, THE BANDS RANGE FROM BAND NUMBER WARPFR TO WARPFR2,
  BETWEEN ROW I1 OF ARY1 AND ROW I2 OF ARY2. E_OR_ZC DETERMINE WHETHER
  THE BANDS'S ENERGY ('E') OR ZERO CROSSING ('Z') VALUE SHOULD BE COMPARED. }

VAR     DIF : INTEGER;           { TEMPORARY STORAGE VARIABLE }
        I : INTEGER;             { LOOP VARIABLE, INDEX INTO FRAMES }
        T1, T2 : TIME_SAMPLE;    { HOLDS ONE FRAME OF DATA--THE FRAMES }
                                 { TO BE COMPARED. }

BEGIN
  IF E_OR_ZC = 'E' THEN BEGIN
    D[I1,I2]:=0;
```

```
    FOR I:=1 TO N_BPF DO BEGIN
      T1[I]:=ARY1[I2,I];
      T2[I]:=ARY2[I1,I];
    END;
    FOR I:= WARPFR TO WARPFR2 DO BEGIN
      DIF:=ABS(T1[I] - T2[I]);
      IF DIF > D[I1,I2] THEN D[I1,I2]:=DIF;
    END;
  END
  ELSE BEGIN
    ZC_D[I1,I2]:=0;
    FOR I:=9 TO 15 DO BEGIN
      T1[I]:=ARY1[I2,I];
      T2[I]:=ARY2[I1,I];
    END;
    FOR I:=WARPFR+8 TO WARPFR2+8 DO BEGIN
      DIF:=ABS(T1[I] - T2[I]);
      IF DIF > ZC_D[I1,I2] THEN ZC_D[I1,I2]:=DIF;
    END;
  END;

END;

{ ------------------------------------------------------------------ }

PROCEDURE GET_D ( X, Y : INTEGER; VAR T3:INTEGER; VAR T4:INTEGER);

{ RETURNS T3=D[X,Y] AND T4=ZC_D[X,Y] IF X,Y ARE VALID VALUES IN THE TIME
  WARPING SCHEME. IF NOT, RETURN WITH T3, T4 = INF_DIST                }

BEGIN
  IF (Y<=Y_MAX[X]) AND (Y>=Y_MIN[X]) THEN BEGIN
    T3:=D[X,Y];
    T4:= ZC_D[X,Y];
  END
  ELSE BEGIN
    T3:=INF_DIST;
    T4:=INF_DIST;
  END;

END;

{ ------------------------------------------------------------------ }

PROCEDURE DP_FCN ( X, Y : INTEGER; E_OR_ZC : CHAR);

{ THIS IS THE DYNAMIC PROGRAMMING FUNCTION.
  IF THE BAND ENERGIES ARE BEING WARPED THEN IT SETS
  D[X,Y] = DIST(X,Y) + MINIMUM( D[X-1,Y], D[X-1,Y-1], D[X-1,Y-2], ... ).
  THE D ENTRIES MUST BE VALID ACCORDING TO Y_MAX[X-1] AND Y_MIN[X-1].
  IF THE BAND ZERO CROSSINGS ARE BEING WARPED THEN IT SETS
  ZC_D[X,Y] = DIST(X,Y) + MINIMUM( ZC_D[X-1,Y], D[X-1,Y-1], .... ).
  THE ZC_D ENTRIES MUST BE VALID ACCORDING TO Y_MAX[X-1] AND Y_MIN[X-1]. }
```

```
VAR     VAL2 : INTEGER;         { IS SET TO THE MINIMUM TIME-WARP SCORE SO FAR }
                                { FOR THESE TWO WORDS IF THE ENERGIES ARE }
                                { BEING WARPED }
        T1 : INTEGER;           { PASSING PARAMETER TO GET_D }
        T2 : INTEGER;           { PASSING PARAMETER TO GET_D }
        T5 : INTEGER;           { PASSING PARAMETER TO GET_D }
        ZC_MIN : INTEGER;       { IS SET TO THE MINIMUM TIME-WARP SCORE SO FAR }
                                { FOR THESE TWO WORDS IF THE ZERO CROSSINGS    }
                                { ARE BEING WARPED. }

BEGIN
  IF E_OR_ZC = 'E' THEN BEGIN
    VAL2:=INF_DIST;
    FOR T5:=Y_MIN[X-1] TO Y_MAX[X-1] DO BEGIN
      IF T5 <= Y THEN BEGIN
        GET_D(X-1,T5,T1,T2);
        IF VAL2 > T1 THEN VAL2:=T1;
      END;
    END;
      IF VAL2 < INF_DIST THEN BEGIN
          DIST(X,Y,WARPFR,WARPFR2,E_OR_ZC);
          D[X,Y]:=D[X,Y]+VAL2;
      END
      ELSE D[X,Y]:=INF_DIST;
  END
  ELSE BEGIN
    ZC_MIN:=INF_DIST;
    FOR T5:=Y_MIN[X-1] TO Y_MAX[X-1] DO BEGIN
      IF T5 <= Y THEN BEGIN
        GET_D(X-1,T5,T1,T2);
        IF ZC_MIN > T2 THEN ZC_MIN:=T2;
      END;
    END;
      IF ZC_MIN < INF_DIST THEN BEGIN
          DIST(X,Y,WARPFR,WARPFR2,E_OR_ZC);
          ZC_D[X,Y]:=ZC_D[X,Y]+ZC_MIN;
      END
      ELSE ZC_D[X,Y]:=INF_DIST;
  END;

END;

{------------------------------------------------------------------------}

PROCEDURE SET_BOUNDS(M_ADJ,A,B : INTEGER);

{ ARRAYS Y_MIN AND Y_MAX ARE SET TO THE  BOUNDS FOR ALLOWED TIME-WARPING.
  Y_MIN[I] AND Y_MAX[I] HOLD THE MINIMUM FRAME NUMBER (LEVEL IN ARRAY
  ARY2) AND THE MAXIMUM FRAME NUMBER WHICH FRAME I OF ARRAY ARY2 MAY BE
  COMPARED AGAINST IN ARRAY ARY1. }

VAR     S : REAL;
```

```
        X : INTEGER;

BEGIN
  S:=(A + M_ADJ) / (B + M_ADJ);
  FOR X:= 1 TO B DO BEGIN
    Y_MIN[X]:=ROUND(S*X) - M_ADJ;
    Y_MAX[X]:=ROUND(S*X) + M_ADJ;
    IF Y_MAX[X] > A THEN Y_MAX[X]:=A;
    IF Y_MIN[X] > A THEN Y_MIN[X]:=A;
    IF Y_MIN[X] < 1 THEN Y_MIN[X]:=1;
  END;

END;

{ ------------------------------------------------------------------ }

BEGIN  { TOTAL_DIST }

  SET_BOUNDS(M_ADJ,A,B);
  FOR Y:=Y_MIN[1] TO Y_MAX[1] DO DIST(1,Y,WARPFR,WARPFR2,E_OR_ZC);
    FOR X:=2 TO B DO
       FOR Y:=Y_MIN[X] TO Y_MAX[X] DO
          DP_FCN(X,Y,E_OR_ZC);
  IF E_OR_ZC = 'E' THEN BEGIN
    VAL:=INF_DIST;
    FOR Y:=Y_MIN[B] TO Y_MAX[B] DO
      IF D[B,Y] < VAL THEN VAL:=D[B,Y];
    IF VAL < INF_DIST THEN VAL:=(VAL + B DIV 2) DIV B;
  END
  ELSE BEGIN
    ZVAL:=INF_DIST;
    FOR Y:=Y_MIN[B] TO Y_MAX[B] DO
      IF ZC_D[B,Y] < ZVAL THEN ZVAL:=ZC_D[B,Y];
    IF ZVAL < INF_DIST THEN ZVAL:=(ZVAL + B DIV 2) DIV B;
  END;

  { RETURN WITH VAL AND ZVAL SET }
END;

{ ------------------------------------------------------------------ }

PROCEDURE GO_RECOGN;

{ GO_RECOGN IS ONE OF THE MAIN DRIVING ROUTINES.  IT READ IN THE COMPARISON
  WORD, AND CALLS PROCEDURE SCORE.  SCORE RETURNS WITH TWO PARAMETERS,
  BEST AND WFLAG, SET AND GO_RECOGN REACTS AS FOLLOWS.
  IF WFLAG = 'A' THEN THE WORD WAS AMBIGUOUS AND BEST IS SET TO THE INDEX
                 OF THE BEST MATCHING WORD.  IF THE BEST MATCHING WORD WAS
                 THIS WORD AND THE LIBRARIAN IS IN THE AUTOMATIC MODE AND
                 LIBRARY WAVE AVERAGING HAS BEEN CHOSEN, THEN THE BEST
                 MATCHING WORD GETS AVERAGED WITH THE COMPARSON WORD.  IF
                 THE BEST MATCHING WORD IS NOT THE SAME AS THE COMPARISON
                 WORD AND THE LIBRAIAN IS IN THE AUTOMATIC MODE (ON) THEN THE
```

```
                COMPARISON WORD IS ADDED TO THE LIBRARY IF THE LIBRARY IS
                NOT FULL.
  IF WFLAG = 'N' THEN THE COMPARISON COULD NOT BE MATCHED AND IS ADDED TO
                THE LIBRARY IF THE LIBRARIAN IS ON AND THE LIBRARY IS NOT FULL.
  IF WFLAG = 'Y' THEN THE COMPARISON WORD WAS MATCHED WITH THE LIBRARY WORD
                POINTED TO BY BEST (BEST BEING AN INDEX INTO THE LIBRARY).
                IF THE BEST MATCHING WORD WAS THE SAME AS THIS WORD, AND THE
                LIBRARIAN IS ON, AND WAVE AVERAGING WAS CHOSEN, THEN THE
                BEST MATCHING WORD AND THE COMPARISON WORD ARE AVERAGED.  IF
                THE BEST MATCHING WORD WAS NOT THIS WORD, AND THE LIBRARIAN
                IS ON, AND THE LIBRARY IS NOT FULL, THEN THIS WORD IS ADDED
                TO THE LIBRARY.     }

TYPE DDIST=ARRAY[1..MAX_VOCAB] OF INTEGER;

VAR TEST      : SAMPLE;              { COMPARISON WORD PATTERN }
    TEST2     : SAMPLE;              { NORMALIZED COMPARISON WORD PATTERN }
    TSTSUM    : ARRAY [1..N_BPF] OF INTEGER;
                                     { SUM OF ALL THE ENERGIES OF EACH BAND }
                                     { FOR THE COMPARISON WORD }
    TOTALTS   : INTEGER;             { TOTAL ENERGY OF THE COMPARISON WORD }
    N         : INTEGER;             { NUMBER OF FRAMES IN THE COMPARISON PATTERN }
    I,W,K,J     : INTEGER;             { LOOP VARIABLES }
    LINE      : LINE_TYPE;           { INPUT LINE BUFFER--HOLDS ONE FRAME }
    BEST      : INTEGER;             { INDEX OF BEST SCORE FOUND }
    SECOND    : INTEGER;             { INDEX OF SECOND BEST SCORE }
    N3        : INTEGER;             { TEMPORARY STORAGE VARIABLE }
    NAME2     : STRING[20];          { TEXT STRING OF COMPARISON WORD }
    NAME3     : STRING[20];          { TEXT STRING CONVERTED TO UPPERCASE }
    CHR       : CHAR;                { TEMPORARY STORAGE VARIABLE }
    MFLAGS    : ARRAY[1..4] OF BYTE; { FLAGS TO DENOTE OF MATCHING }
                                     { ALGORITHMS HAVE BEEN ENTERED }
    VER_FLAG  : BYTE;                { FLAG TO DENOTE IF THE VERIFY }
                                     { PROCEDURE HAS BEEN ENTERED }
    VSUM      : BYTE;                { MATCH3 VERIFY SCORE }
    WFLAG     : CHAR;                { FLAG SET AS DESCRIBED EARLIER }
    VFLAG     : CHAR;                { FLAG TO DENOTE IF VERIFY WAS }
                                     { SUCCESSFULL }
    DMNS      : INTEGER;             { DIFFERENCE BETWEEN BEST AND SECOND }
                                     { BEST WORDS AT VARIOUS STAGES }
    BEST1,BEST2,BEST3,BEST4 : INTEGER; { INDEXES OF BEST WORDS AT THE }
                                       { CORRESPONDING STAGES }
    SECOND1,SECOND2,SECOND3,SECOND4 : INTEGER; { SECOND BEST WORDS AT }
                                               { CORRESPONDING STAGES }
    DIST1,DIST2,DIST3,DIST4 : DDIST;   { ARRAY WHICH HOLD ALL SCORES AT }
                                       { THE CORRESPONDING STAGES }
    DIST1_2,DIST2_3         : DDIST;   { ARRAYS WHICH HOLD ACCUMILATED SCORES }
                                       { AT CORRESPONDING STAGES }

{PROCEDURE A872(VAR LINE:LINE_TYPE);EXTERNAL 'A872.COM';}
{ ASSEMBLY ROUTINE TO INTERFACE WITH THE F-E BOX. THE ROUTINE MUST BE }
{ CALLED WITH AN ARRAY AS THE ONLY VARIABLE BEING PASSED. THE ROUTINE WILL }
{ SEND OUT A NL TO THE F-E BOX AND THEN READ INTO THE PASSED ARRAY THE }
```

```
  { DATA THAT THE F-E BOX SENDS BACK UNTIL EITHER A NL OR CR IS SENT BY THE }
  { F-E BOX.  THE NL OR CR IS THE LAST CHARACTER  PLACED IN THE ARRAY }
  { THIS ROUTINE IS NOT USED IN RMATCHS AT THE MOMENT, SINCE ALL THE }
  { COMPARISON WORDS ARE READ FROM FILES SO AS TO PRODUCE REPEATABILITY OF }
  { TESTING }

  { ---------------------------------------------------------------------- }

PROCEDURE SCORE (VAR BEST : INTEGER; VAR WFLAG : CHAR );

{ BY USING THE DIFFERENT MATCHING ALGORITHMS AND CRITERIA, PROCEDURE SCORE
  FINDS THE BEST MATCHING LIBRARY WORD, OR A NO MATCH CONDITION.  THE
  VARIABLES THAT SCORE RETURNS ARE DESCRIBED IN PROCEDURE GO_RECOGN }

VAR     VAL     : INTEGER;  { HOLDS TIME WARP SCORES FOR ENERGIES }
        ZVAL    : INTEGER;  { HOLDS TIME WARP SCORES FOR ZERO-XINGS }
        K,J,J2  : INTEGER;  { INDEX VARIABLES }

{---------------------------------------------------------------------- }

PROCEDURE SUMTEST;

{ SUMS UP THE ENERGIES IN EACH BAND OF THE COMPARISON WORD AND STORES
  THESE SEVEN VALUES IN ARRAY TSTSUM.  SETS TOTALTS TO THE SUM OF ALL
  THE ENERGY IN THE COMPARISON WORD }

VAR J,K : INTEGER;

BEGIN
  TOTALTS:=0;
  FOR J:= 1 TO N_BPF DO BEGIN
    TSTSUM[J]:=0;
    FOR K:=1 TO N DO TSTSUM[J]:=TSTSUM[J] + TEST[K,J];
    TOTALTS:= TOTALTS + TSTSUM[J];
  END;
END;

{ ---------------------------------------------------------------------- }

PROCEDURE MATCHI (VAR DISTI:DDIST; FRM_THR: INTEGER);

{ RETURNS WITH DISTI[I] EQUAL TO THE SUM OF THE DIFFERENCES OF THE AVERAGE
  ENERGIES IN EACH BAND BETWEEN LIBRARY WORD I AND THE COMPARISON WORD.
  BEFORE CALCULATING THIS SUM, THE TWO PATTERNS ARE NORMALIZED (IN PROCEDURE
  ENG_DIF).  DISTI] = 255 (HIGHEST POSSIBLE SCORE) IF THE LIBRARY WORD I AND
  THE COMPARISON WORD DIFFER BY MORE THAN FRM_THR NUMBER OF FRAMES IN LENGTH }

VAR     ENSDIF,I : INTEGER;

PROCEDURE ENG_DIF(S:INTEGER; VAR ENSDIF: INTEGER);

VAR     TEMP,RR2 : REAL;
        J,TEMP2  : INTEGER;
```

```
        TEMPSUM  : ARRAY[1..N_BPF] OF INTEGER;
        TSSUM2   : ARRAY[1..N_BPF] OF INTEGER;

BEGIN
  TEMP2:=ROUND((WDSUM[S]-BDSUM[S,7])/WORDS[S].N);
  TEMP:=ROUND((TOTALTS-TSTSUM[7])/N)-TEMP2;
  FOR J:=1 TO N_BPF-1 DO BEGIN
    RR2:=ROUND(BDSUM[S,J]/WORDS[S].N)/TEMP2;
    TEMPSUM[J]:=ROUND(BDSUM[S,J]/WORDS[S].N)+ROUND(TEMP*RR2);
  END;
  FOR J:=1 TO 6 DO TSSUM2[J]:=ROUND(TSTSUM[J]/N);
  ENSDIF:=0;
  FOR J:=1 TO 6 DO ENSDIF:=ENSDIF + ABS(TEMPSUM[J] - TSSUM2[J]);

END;

{ ------------------------------------------------------------------- }

BEGIN   { PROCEDURE MATCH1 }

  MFLAGS[I]:=1;
  FOR I:= 1 TO N_WORDS DO BEGIN
    IF ABS(WORDS[I].N -N) > FRM_THR THEN DIST1[I]:=HSCORE
    ELSE BEGIN
      ENG_DIF(I,ENSDIF);
      DIST1[I]:=ENSDIF;
      {WRITELN('WORDS[I].TXT   ',DIST1[I]);}
    END;
  END;

END;

{ ------------------------------------------------------------------- }

PROCEDURE MATCH2 (VAR DIST2:DDIST; DIST1:DDIST; THR1:INTEGER; S_OR_IC:CHAR);

{ SETS DIST2[I] EQUAL TO THE TIME-WARP SCORE FOR ALL ENEROY BANDS COMBINED
  FOR LIBRARY WORD I AND THE COMPARISON WORD.  IF THE LIBRARY WORD I HAD A
  STAGE 1 (DIST1) SCORE ABOVE THRESHOLD 1 (THR1) THEN DIST2[I]=255 (HIGH
  SCORE) WITHOUT PERFORMING THE TIME-WARPING, I.E., LIBRARY WORD I IS NOT
  CONSIDERED TO BE A POSSIBLE MATCH.  BEFORE PERFORMING THE TIME WARPING
  THE COMPARISON WORD IS TIME-WARPED TO THE LIBRARY WORD. }


VAR     I : INTEGER;

PROCEDURE NORMALIZE(I: INTEGER);

VAR     RR2,RR,DF,TD : INTEGER;
        N_TEST       : ARRAY[1..N_BPF] OF REAL;
        J,K          : INTEGER;
        RR3          : REAL;
```

```
BEGIN
  RR2:=ROUND(TOTALTS/N);
  RR:=ROUND(WDSUM[I]/WORDS[I].N);
  DF:=RR-RR2;
  FOR J:=1 TO N_BPF DO BEGIN
    RR3:=TSTSUM[J]/TOTALTS;
    N_TEST[J]:=DF*RR3*N;
  END;
  FOR K:=1 TO N_BPF DO BEGIN
    IF N_TEST[K] <> 0 THEN BEGIN
      FOR J:=1 TO N DO BEGIN
        RR3:=TEST[J,K]/TSTSUM[K];
        TD:=ROUND(N_TEST[K]*RR3);
        TEST2[J,K]:=TEST[J,K]+TD;
        IF TEST2[J,K] > 255 THEN TEST2[J,K]:=255;
        IF TEST2[J,K] < 0 THEN TEST2[J,K]:=0;
      END;
    END
    ELSE FOR J:=1 TO N DO TEST2[J,K]:=TEST[J,K];
  END;

END;

{ ------------------------------------------------------------------ }


BEGIN   { PROCEDURE MATCH2 }
  MFLAGS[2]:=1;
  WARPFR:=1;
  WARPFR2:=7;
  E_OR_ZC:='E';
  FOR I:= 1 TO N_WORDS DO BEGIN
    IF DIST1[I] > THR1 THEN DIST2[I]:=HSCORE
    ELSE BEGIN
      NORMALIZE(I);
      IF WORDS[I].N < N
      THEN TOTAL_DIST(TEST2,N,WORDS[I].DAT,WORDS[I].N,VAL,ZVAL,E_OR_ZC)
      ELSE TOTAL_DIST(WORDS[I].DAT,WORDS[I].N,TEST2,N,VAL,ZVAL,E_OR_ZC);
      DIST2[I]:=VAL;
    END;
  END;

END;

{ ------------------------------------------------------------------ }

PROCEDURE MATCH3(VAR DIST3:DDIST; DIST1_2:DDIST);

{ SETS DIST3[I] EQUAL TO THE SUM OF THE TIME-WARP SCORES FOR ZERO CROSSING
  BANDS 5,6, AND 7 (WARP INDIVIDUALLY )_
  BETWEEN LIBRARY WORD I AND THE COMPARISON WORD.
  IF THE LIBRARY WORD WAS DETERMINED BY MATCH1 AND/OR MATCH2 TO LONGER
  BE CONSIDERED AS A MATCHING WORD, THEN DIST1_2[I] WILL EQUAL THE HIGH
```

```
  SCORE (255), AND MATCH3 WILL SET DIST3[I] EQUAL TO THE HIGH SCORE. }

VAR     I,J : INTEGER;

BEGIN
  MFLAGS[3]:=1;
  E_OR_IC:='I';
  FOR I:=1 TO N_WORDS DO BEGIN
    IF DIST1_2[I] = HSCORE THEN BEGIN
      DIST3[I]:=HSCORE;
    END
    ELSE BEGIN
      DIST3[I]:=0;
      FOR J:=1 TO 3 DO BEGIN
        WARPFR:=J;
        WARPFR2:=J;
        IF WORDS[I].N < N
        THEN TOTAL_DIST(TEST,N,WORDS[I].DAT,WORDS[I].N,VAL,ZVAL,E_OR_IC)
        ELSE TOTAL_DIST(WORDS[I].DAT,WORDS[I].N,TEST,N,VAL,ZVAL,E_OR_IC);
        DIST3[I]:=DIST3[I] + ZVAL;
      END;
    END;
  END;

END;

{ ---------------------------------------------------------- }

PROCEDURE MATCH4 (VAR DIST4 : DDIST; DIST2_3 : DDIST);

{ SETS DIST4[I] EQUAL TO THE TIME-WARP SCORES THE ENERGY READINGS IN
  BAND 1 BETWEEN LIBRARY WORD I AND THE COMPARISON WORD.
  IF THE LIBRARY WORD IS NO LONGER BEING CONSIDERED
  AS A MATCHING WORD, THEN DIST2_3[I] WILL EQUAL THE HIGH
  SCORE (255), AND MATCH4 WILL SET DIST4[I] EQUAL TO THE HIGH SCORE. }

VAR     I : INTEGER;

BEGIN
  MFLAGS[4]:=1;
  E_OR_IC:='E';
  WARPFR:=1;
  WARPFR2:=1;
  FOR I:=1 TO N_WORDS DO BEGIN
    IF DIST2_3[I] = HSCORE THEN BEGIN
      DIST4[I]:=HSCORE;
    END
    ELSE BEGIN
      IF WORDS[I].N < N
        THEN TOTAL_DIST(TEST2,N,WORDS[I].DAT,WORDS[I].N,VAL,ZVAL,E_OR_IC)
        ELSE TOTAL_DIST(WORDS[I].DAT,WORDS[I].N,TEST2,N,VAL,ZVAL,E_OR_IC);
      DIST4[I]:=VAL;
    END;
```

```
  END;

END;

{ ------------------------------------------------------------------ }

PROCEDURE COMBINE(VECA:DDIST; VECB:DDIST; VAR VECAB:DDIST; THRB:INTEGER);

{ THIS PROCEDURE SUMS THE TWO VECTORS VECA AND VECB TO FORM VECAB. IF
  VECB[I] IS GREATER THAN THRB THEN VECAB IS SET EQUAL TO THE HIGH SCORE }

VAR     I : INTEGER;

BEGIN
  FOR I:=1 TO N_WORDS DO
    IF VECB[I] > THRB THEN VECAB[I]:=HSCORE
    ELSE VECAB[I]:=VECA[I] + VECB[I];

END;

{ ------------------------------------------------------------------ }

PROCEDURE FNDBST(VEC:DDIST; VAR LW : INTEGER; VAR SLW :INTEGER;
                 VAR DIFF : INTEGER);

{ FINDS THE LOWEST SCORE IN ARRAY DDIST.  IT SETS LW TO THE INDEX OF
  THE LOWEST SCORE, AND SLW TO THE INDEX OF THE SECOND LOWEST SCORE.
  VARIABLE DIFF IS SET TO THE DIFFERENCE BETWEEN THE TWO LOWEST SCORES. }

VAR     I : INTEGER;

BEGIN
  LW:=1;
  SLW:=2;
  IF VEC[1] > VEC[2] THEN BEGIN
    LW:=2;
    SLW:=1;
  END;
  FOR I:=3 TO N_WORDS DO BEGIN
    IF VEC[I] < VEC[SLW] THEN
      IF VEC[I] < VEC[LW] THEN BEGIN
        IF WORDS[I].TXT = WORDS[LW].TXT THEN LW:=I
        ELSE BEGIN
          SLW:=LW;
          LW:=I;
        END;
      END
      ELSE IF WORDS[I].TXT <> WORDS[LW].TXT THEN SLW:=I;
  END;
  DIFF:=VEC[SLW]-VEC[LW];
END;

{ ------------------------------------------------------------------ }
```

```
PROCEDURE VERIFY(X:INTEGER; VAR VFLAG : CHAR; THR3 : INTEGER);

{ THIS PROCEDURE IS CALLED IF THE BEST MATCHING WORD AFTER STAGE 2 HAS
  A COMBINED SCORE WHICH IS LOWER BY A CERTAIN THRESHOLD (REC_DIF2) THAN
  THE SECOND LOWEST SCORE.  THIS PROCEDURE THEN CHECKS ONLY THIS WORD
  WITH THE MATCH3 TEST--SUM OF THE TIME WARP VALUES OF THE ZERO CROSSINGS
  OF BANDS 5,6, AND 7.  IF THIS SUM IS LESS THAN OR EQUAL TO THR3 THEN VFLAG
  IS SET TO "Y", ELSE VFLAG IS SET TO "N". VARIABLE X HOLDS THE INDEX INTO THE
  LIBRARY WHERE THIS BEST MATCHING WORD IS LOCATED. }

VAR     J : INTEGER;

BEGIN
  E_OR_ZC:='Z';
  VER_FLAG:=1;
  VSUM:=0;
  FOR J:=1 TO 3 DO BEGIN
    WARPFR:=J;
    WARPFR2:=J;
    IF WORDS[X].N < N
      THEN TOTAL_DIST(TEST,N,WORDS[X].DAT,WORDS[X].N,VAL,ZVAL,E_OR_ZC)
      ELSE TOTAL_DIST(WORDS[X].DAT,WORDS[X].N,TEST,N,VAL,ZVAL,E_OR_ZC);
    VSUM:=VSUM + ZVAL;
  END;
  IF VSUM <= THR3 THEN VFLAG:='Y'
  ELSE VFLAG:='N';

END;

{ ---------------------------------------------------------------- }

BEGIN   { PROCEDURE SCORE }
  SUMTEST;
  FOR I:=1 TO 4 DO MFLAGS[I]:=0;
  VER_FLAG:=0;
  WRITELN('----------------------------------------------------------');
  WRITELN('CALLING MATCH1');
  MATCH1(DIST1,FRM_THR);
  WRITELN('CALLING MATCH2');
  MATCH2(DIST2,DIST1,THR1,E_OR_ZC);
  COMBINE(DIST1,DIST2,DIST1_2,THR2);
  FNDBST(DIST1_2,BEST2,SECOND2,BMNS);
  VFLAG:='N';
  IF (DIST1_2[BEST2] <> HSCORE) AND (BMNS >= REC_DIF2) THEN BEGIN
    WRITELN('CALLING VERIFY');
    VERIFY(BEST2,VFLAG,THR3);
  END;
  IF VFLAG='Y' THEN BEGIN
    BEST:=BEST2;
    WFLAG:='Y';
  END
  ELSE BEGIN
```

```
    IF DIST1_2[BEST2] <> HSCORE THEN BEGIN
      WRITELN('CALLING MATCH3');
      MATCH3(DIST3,DIST1_2);
      COMBINE(DIST1_2,DIST3,DIST2_3,THR3);
      FNDBST(DIST2_3,BEST3,SECOND3,BMNS);
      BEST:=BEST3;
      IF (DIST2_3[BEST3] <> HSCORE) AND (BMNS >= REC_DIF3) THEN BEGIN
        WFLAG:='Y';
      END
      ELSE BEGIN
        IF DIST2_3[BEST3] <> HSCORE THEN BEGIN
          WRITELN('CALLING MATCH4');
          MATCH4(DIST4,DIST2_3);
          FNDBST(DIST4,BEST4,SECOND4,BMNS);
          IF (DIST4[BEST4] <= THR4) AND (BMNS >= REC_DIF4) THEN BEGIN
            BEST:=BEST4;
            WFLAG:='Y';
          END
          ELSE WFLAG:='A';
        END
        ELSE WFLAG:='N';
      END;
    END
    ELSE WFLAG:='N';
  END;

END;

{ ---------------------------------------------------------- }

PROCEDURE WLOG;

{ WRITES ONTO THE SCREEN THE TWO LOWEST SCORING WORDS AFTER EACH STAGE
  AND WHETHER THE COMPARISON WAS FOUND TO MATCH CORRECTLY OR INCORRECTLY,
  OR IF IT WAS FOUND TO BE AMBIGUOUS, OR IT WAS FOUND TO NOT MATCH AT ALL }

VAR DIFF : INTEGER;

PROCEDURE HEADER2;

BEGIN
  WRITELN;
  WRITELN('PROGRAM RMATCH5');
  WRITELN('LIBRARY FILE - ',VOCAB_FILE,'  COMP. FILE - ',FILENAME2);
  WRITELN('COMPARISON WORD - ',NAME2,'   INDEX - ',WIDX);
  WRITELN;
END;

{ ---------------------------------------------------------- }

BEGIN
  HEADER2;
  IF MFLAGS[1]=1 THEN BEGIN
```

```
    BEST1:=1;
    SECOND1:=2;
    IF DIST1[1] > DIST1[2] THEN BEGIN
      BEST1:=2;
      SECOND1:=1;
    END;
    FOR I:=3 TO N_WORDS DO BEGIN
      IF DIST1[I] < DIST1[SECOND1] THEN
        IF DIST1[I] < DIST1[BEST1] THEN BEGIN
          IF WORDS[I].TXT = WORDS[BEST1].TXT THEN BEST1:=I
          ELSE BEGIN
            SECOND1:=BEST1;
            BEST1:=I;
          END;
        END
        ELSE IF WORDS[I].TXT <> WORDS[BEST1].TXT THEN SECOND1:=I;
    END;
    WRITE('STAGE 1:   ',WORDS[BEST1].TXT,' (',BEST1,')',' - ',DIST1[BEST1]);
    WRITELN('       ',WORDS[SECOND1].TXT,' (',SECOND1,')',' - ',DIST1[SECOND1]);
  END;
  IF MFLAGS[2]=1 THEN BEGIN
    WRITE('STAGE 2:   ',WORDS[BEST2].TXT,' (',BEST2,')',' - ',DIST1_2[BEST2]);
    WRITELN('       ',WORDS[SECOND2].TXT,' (',SECOND2,')',' - ',DIST1_2[SECOND2]);
  END;
  IF VER_FLAG=1 THEN BEGIN
    IF VFLAG = 'Y'
      THEN WRITELN('VERIFY GOOD:  STAGE 3 SCORE - ',VSUM)
      ELSE WRITELN('VERIFY NOT GOOD:  STAGE 3 SCORE - ',VSUM);
  END;
  IF MFLAGS[3]=1 THEN BEGIN
    WRITE('STAGE 3:   ',WORDS[BEST3].TXT,' (',BEST3,')',' - ',DIST2_3[BEST3]);
    WRITELN('       ',WORDS[SECOND3].TXT,' (',SECOND3,')',' - ',DIST2_3[SECOND3]);
  END;
  IF MFLAGS[4]=1 THEN BEGIN
    WRITE('STAGE 4:   ',WORDS[BEST4].TXT,' (',BEST4,')',' - ',DIST4[BEST4]);
    WRITELN('       ',WORDS[SECOND4].TXT,' (',SECOND4,')',' - ',DIST4[SECOND4]);
  END;
  IF WFLAG = 'N' THEN WRITELN('NO MATCH');
  IF WFLAG = 'A' THEN WRITELN('AMBIGUOUS');
  IF WFLAG = 'Y' THEN WRITELN('MATCHED WITH WORD ',WORDS[BEST].TXT);
  WRITELN;

END;

{ ---------------------------------------------------------------- }

PROCEDURE WLOGLST;

{ WRITES ONTO THE LST DEVICE THE TWO LOWEST SCORING WORDS AFTER EACH STAGE
  AND WHETHER THE COMPARISON WAS FOUND TO MATCH CORRECTLY OR INCORRECTLY,
  OR IT WAS FOUND TO BE AMBIGUOUS, OR IT WAS FOUND TO NOT MATCH AT ALL }

PROCEDURE HEADER3;
```

```
BEGIN
  {WRITELN(LST,'PROGRAM RMATCH5');  }
  WRITELN(LST,'LIBRARY FILE - ',VOCAB_FILE,'  COMP. FILE - ',FILENAME2);
  WRITELN(LST,'COMPARISON WORD - ',NAME2,'   INDEX - ',WIDX);
  {WRITELN(LST);                    }
END;

{ ------------------------------------------------------------ }

BEGIN
  HEADER3;
  IF MFLAGS[1]=1 THEN BEGIN
    WRITE(LST,'STAGE 1:   ',WORDS[BEST1].TXT,' (',BEST1,')',' -  ',DIST1[BEST1]);
    WRITELN(LST,'       ',WORDS[SECOND1].TXT,' (',SECOND1,')',' -  ',DIST1[SECOND1]);
  END;
  IF MFLAGS[2]=1 THEN BEGIN
    WRITE(LST,'STAGE 2:   ',WORDS[BEST2].TXT,' (',BEST2,')',' -  ',DIST1_2[BEST2]);
    WRITELN(LST,'       ',WORDS[SECOND2].TXT,' (',SECOND2,')',' -  ',DIST1_2[SECOND2]);
  END;
  IF VER_FLAG=1 THEN BEGIN
    IF VFLAG = 'Y'
      THEN WRITELN(LST,'VERIFY GOOD;  STAGE 3 SCORE - ',VSUM)
      ELSE WRITELN(LST,'VERIFY NOT GOOD;  STAGE 3 SCORE - ',VSUM);
  END;
  IF MFLAGS[3]=1 THEN BEGIN
    WRITE(LST,'STAGE 3:   ',WORDS[BEST3].TXT,' (',BEST3,')',' -  ',DIST2_3[BEST3]);
    WRITELN(LST,'       ',WORDS[SECOND3].TXT,' (',SECOND3,')',' -  ',DIST2_3[SECOND3]);
  END;
  IF MFLAGS[4]=1 THEN BEGIN
    WRITE(LST,'STAGE 4:   ',WORDS[BEST4].TXT,' (',BEST4,')',' -  ',DIST4[BEST4]);
    WRITELN(LST,'       ',WORDS[SECOND4].TXT,' (',SECOND4,')',' -  ',DIST4[SECOND4]);
  END;
  {WRITELN(LST);}
  IF WFLAG = 'N' THEN BEGIN
    WRITELN(LST,'NO MATCH','N':49);
    IF CMD1='A' THEN
      IF FULLFLAG2 <> 1
        THEN WRITELN(LST,'THE WORD WAS ADDED TO THE LIBRARY, INDEX - ',N_WORDS)
        ELSE WRITELN(LST,'LIBRARY IS FULL - WORD COULD NOT BE ADDED. ');
  END;
  IF WFLAG = 'A' THEN BEGIN
    WRITELN(LST,'AMBIGUOUS','A':49);
    IF (CMD1 = 'A') AND (NAME2 = WORDS[BEST].TXT) AND (CMD16='Y')
      THEN BEGIN
        WRITE(LST,'WORD WAS AVERAGED WITH ',WORDS[BEST].TXT);
        WRITELN(LST,'    INDEX INTO LIB. - ',BEST);
      END;
    IF (CMD1 = 'A') AND (NAME2 <> WORDS[BEST].TXT)
      THEN IF FULLFLAG2 <> 1
        THEN WRITELN(LST,'WORD WAS ADDED TO THE LIBRARY, INDEX - ',N_WORDS)
        ELSE WRITELN(LST,'LIBRARY IS FULL - WORD COULD NOT BE ADDED. ');
  END;
```

```
  IF WFLAG = 'Y' THEN BEGIN
    IF CMD1 = 'A' THEN BEGIN
      IF NAME2 = WORDS[BEST].TXT THEN BEGIN
        WRITELN(LST,'THE WORD WAS MATCHED CORRECTLY','R':27);
        IF CMD16='Y' THEN
          WRITE(LST,'THE WORD WAS AVERAGED WITH ',WORDS[BEST].TXT);
        WRITELN(LST,'   INDEX INTO LIB. - ',BEST);
      END
      ELSE BEGIN
       WRITELN(LST,'THE WORD WAS MATCHED INCORRECTLY','W':25);
       IF FULLFLAG2 <> 1
         THEN WRITELN(LST,'IT WAS ADDED TO THE LIBRARY, INDEX - ',N_WORDS)
         ELSE WRITELN(LST,'LIBRARY IS FULL - WORD COULD NOT BE ADDED. ');
     END;
    END;
  END;
  IF FULLFLAG = 1 THEN FULLFLAG2:=1;
  WRITELN(LST,'-------------------------------------------------------------------------------');

END;

{------------------------------------------------------------------ }
PROCEDURE WAVEAVE(BEST:INTEGER);

{ THIS PROCEDURE AVERAGES THE WORD PATTERN IN ARRAY TEST WITH THE WORD
  IN THE LIBRARY PASSED GIVEN BY THE INDEX PASSED IN BEST.  THE AVERAGING
  IS A WEIGHTED AVERAGE, WITH THE LIBRARY WORD GIVEN A WEIGHT OF NUM-1 AND
  THE COMPARISON WORD A WEIGHT OF 1.  THE SMALLER, FRAMEWISE, OF THE TWO
  WORDS IS 'CENTERED' UPON THE LARGER WORD BEFORE AVERAGING.  THE FINAL
  WORD THUS IS THE SIZE OF THE LARGER OF THE TWO WORDS.  THE FRAMES IN THE
  LARGER WORD WHICH DO NOT HAVE CORRESPONDING FRAMES IN THE SMALLER WORD
  (THE FRAMES AT EACH END) ARE LEFT AS IS.  NORMALIZING IS DONE BEFORE
  AVERAGING }

VAR DIFFQ:INTEGER;
    SHIFT : INTEGER;
    I,J   : INTEGER;

PROCEDURE NORMALIZE3(NDX:INTEGER);

VAR     RR,RR2,DF : INTEGER;
        J,K,TD    : INTEGER;
        RR3       : REAL;
        N_TEST    : ARRAY[1..N_BPF] OF REAL;

BEGIN

  RR2:=ROUND(TOTALTS/N);
  RR:=ROUND(WDSUM[NDX]/WORDS[NDX].N);
  DF:=RR-RR2;
  FOR J:=1 TO N_BPF DO BEGIN
    RR3:=TSTSUM[J]/TOTALTS;
    N_TEST[J]:=DF*RR3*N;
```

```
  END;
  FOR K:=1 TO N_BPF DO BEGIN
    IF N_TEST[K] <> 0 THEN BEGIN
      FOR J:=1 TO N DO BEGIN
        RR3:=TEST[J,K]/TSTSUM[K];
        TD:=ROUND(N_TEST[K]*RR3);
        TEST2[J,K]:=TEST[J,K]+TD;
        IF TEST2[J,K] > 255 THEN TEST2[J,K]:=255;
        IF TEST2[J,K] < 0 THEN TEST2[J,K]:=0;
      END;
    END
    ELSE FOR J:=1 TO N DO TEST2[J,K]:=TEST[J,K];
  END;
  FOR K:=1 TO N DO
    FOR J:=8 TO 15 DO
      TEST2[K,J]:=TEST[K,J];

END;

{ ------------------------------------------------------------------ }

BEGIN  { WAVEAVE }

  WRITELN('AVERAGE WITH WORD ',WORDS[BEST].TXT,'  INDEX-',BEST);
  NORMALIZE3(BEST);
  RIGHTON:=1;
  SHIFT:=WORDS[BEST].N-N;
  IF SHIFT=0 THEN BEGIN
    DIFFQ:=1;
    FOR I:=DIFFQ TO WORDS[BEST].N DO
      FOR J:=1 TO N_ELM DO
        WORDS[BEST].DAT[I,J]:=ROUND(((NUM-1)*WORDS[BEST].DAT[I,J]+TEST2[I,J])/NUM);
  END;
  IF SHIFT > 0 THEN BEGIN
    DIFFQ:=ROUND((WORDS[BEST].N-N)/2);
    IF ODD(DIFFQ) THEN BEGIN
      FOR I:= DIFFQ+1 TO WORDS[BEST].N-DIFFQ DO
        FOR J:=1 TO N_ELM DO
          WORDS[BEST].DAT[I,J]:=ROUND(((NUM-1)*WORDS[BEST].DAT[I,J]+TEST2[I-DIFFQ,J])/NUM);
    END
    ELSE BEGIN
      FOR I:=DIFFQ+1 TO WORDS[BEST].N-DIFFQ DO
        FOR J:= 1 TO N_ELM DO
          WORDS[BEST].DAT[I,J]:=ROUND(((NUM-1)*WORDS[BEST].DAT[I,J]+TEST2[I-DIFFQ,J])/NUM);
    END;
  END;
  IF SHIFT < 0 THEN BEGIN
    DIFFQ:=ROUND((N-WORDS[BEST].N)/2);
    IF ODD(DIFFQ) THEN BEGIN
      FOR I:=DIFFQ+1 TO N-DIFFQ DO
        FOR J:=1 TO N_ELM DO
          TEST2[I,J]:=ROUND(((NUM-1)*WORDS[BEST].DAT[I-DIFFQ,J]+TEST2[I,J])/NUM);
    END
```

```
    ELSE BEGIN
      FOR I:=DIFF0+1 TO N-DIFF0 DO
        FOR J:=1 TO N_ELM DO
          TEST2[I,J]:=ROUND(((NUM-1)*WORDS[BEST].DAT[I-DIFF0,J]+TEST2[I,J])/NUM);
    END;
    FOR I:= 1 TO N DO
      FOR J:=1 TO N_ELM DO
        WORDS[BEST].DAT[I,J]:=TEST2[I,J];
    WORDS[BEST].N:=N;
  END;
END;

{ ---------------------------------------------------------- }

PROCEDURE EXPANDOR;

{ THIS PROCEDURE ADDS THE WORD PATTERN IN ARRAY TEST AS A NEW WORD AT THE
  END OF THE LIBRARY }

VAR     I : INTEGER;
        J : INTEGER;

BEGIN

  IF (CMD1 = 'U') AND (N_WORDS >= MAX_VOCAB) THEN BEGIN
    WRITELN('LIBRARY IS FULL - WORD CANNOT BE ADDED');
  END
  ELSE BEGIN
    RIGHTON:=1;
    N_WORDS:=N_WORDS + 1;
    FOR I:=1 TO N DO
      FOR J:=1 TO N_ELM DO
        WORDS[N_WORDS].DAT[I,J]:=TEST[I,J];
    WORDS[N_WORDS].N:=N;
    WORDS[N_WORDS].TXT:=NAME2;
    FOR J:=1 TO N_BPF DO BDSUM[N_WORDS,J]:=TSTSUM[J];
    WDSUM[N_WORDS]:=TOTALTS;
    IF N_WORDS = MAX_VOCAB THEN FULLFLAG:=1;
  END;

END;

{ ---------------------------------------------------------- }

PROCEDURE BANDAVE(VAR ARY:SAMPLE; NF:INTEGER);

{ THIS ROUTINE AVERAGES THE INCOMING COMPARISON WAVE BEFORE MATCHING IS
  DONE.  IT DOES THIS AVERAGING BY SETTING THE ENERGY IN BAND X FRAME Y
  EQUAL TO THE AVERAGE ENERGY IN BAND X FRAME Y AND BAND X FRAME Y-1.
  THE FIRST FRAME IS UNTOUCHED. }

VAR     I,J : INTEGER;
```

```
BEGIN
  FOR I:=1 TO N_BPF DO
    FOR J:=2 TO NF DO
      ARY[J,I]:=ROUND((ARY[J-1,I]+ARY[J,I])/2);
END;

{ ---------------------------------------------------------------------- }

BEGIN  { GO_RECOGN }
  RESET(VOCAB);
  WIDX:=WIDX+1;
  READLN(VOCAB2,NAME2);
  NAMED:='';
  FOR K:=1 TO LENGTH(NAME2) DO BEGIN
    CHR:=COPY(NAME2,K,1);
    CHR:=UPCASE(CHR);
    NAMED:=CONCAT(NAMED,CHR);
  END;
  NAME2:=NAMED;
  READLINE(VOCAB2,LINE);
  N3:=0;
  WHILE LINE[1] <> '*' DO BEGIN
    N3:=N3+1;
    CONVERT(LINE,TEST,N3,MNSFLAG);
    IF MNSFLAG = 1 THEN N3:=N3-1;
    READLINE(VOCAB2,LINE);
  END;
  READLINE(VOCAB2,LINE);
  WHILE LINE[1] <> '$' DO BEGIN
    N3:=N3+1;
    CONVERT(LINE,TEST,N3,MNSFLAG);
    IF MNSFLAG = 1 THEN N3:=N3+1;
    READLINE(VOCAB2,LINE);
  END;
  N:=N3;
  IF CMD3 = 'Y' THEN BANDAVE(TEST,N);
  SCORE(BEST,WFLAG);
  WLOG;
  IF CMD1 ='A' THEN BEGIN
    IF WFLAG='Y' THEN BEGIN
      IF NAME2=WORDS[BEST].TXT THEN BEGIN
        IF CMD16='Y' THEN BEGIN
          WRITELN('WORD WAS MATCHED CORRECTLY AND WILL BE AVERAGED.');
          WAVEAVE(BEST);
        END
        ELSE WRITELN('WORD WAS MATCHED CORRECTLY');
      END
      ELSE BEGIN
        IF FULLFLAG <> 1 THEN BEGIN
          WRITELN('WORD WAS MATCHED INCORRECTLY AND WILL BE ADDED TO THE LIBRARY.');
          EXPANDOR;
        END
        ELSE WRITELN('WORD WAS MATCHED INCORRECTLY. LIBRARY IS FULL - WORD NOT ADDED.');
```

```
      END;
    END;
    IF WFLAG='N' THEN BEGIN
      IF FULLFLAG <> 1 THEN BEGIN
      WRITELN('WORD COULD NOT BE MATCHED AND WILL BE ADDED TO THE LIBRARY.');
      EXPANDOR;
      END
      ELSE WRITELN('WORD COULD NOT BE MATCHED. LIBRARY IS FULL - WORD NOT ADDED.');
    END;
    IF WFLAG='A' THEN BEGIN
      IF NAME2 = WORDS[BEST].TXT THEN BEGIN
        WRITELN('THE WORD WAS AMBIGUOUS, BUT THE BEST MATCHING WORD WAS THIS WORD.');
        IF CMD16='Y' THEN BEGIN
          WRITELN('THEREFORE IT WILL BE AVERAGED WITH THE LIBRARY WORD');
          WAVEAVE(BEST);
        END;
      END
      ELSE BEGIN
        WRITELN('THE WORD WAS AMBIGUOUS AND THE BEST MATCHING WORD WAS NOT THIS WORD');
        IF FULLFLAG <> 1 THEN BEGIN
          WRITELN('THEREFORE IT WILL BE ADDED TO THE LIBRARY');
          EXPANDOR;
        END
        ELSE WRITELN('LIBRARY IS FULL - WORD NOT ADDED. ');
      END;
    END;
  END;
  IF CMD15 = 'Y' THEN WLOGLST;

END;

{ ---------------------------------------------------------------------- }


PROCEDURE DISPLAY_PARAMS;

{ DISPLAYS THE ADJUSTABLE PARAMETERS }

  BEGIN
    WRITELN('(1) FRAME THRESHOLD:          ',FRM_THR);
    WRITELN('(2) STAGE 1 THRESHOLD:        ',THR1);
    WRITELN('(3) STAGE 2 THRESHOLD:        ',THR2);
    WRITELN('(4) STAGE 2 DIFFERENCE:       ',REC_DIF2);
    WRITELN('(5) STAGE 3 THRESHOLD:        ',THR3);
    WRITELN('(6) STAGE 3 DIFFERENCE:       ',REC_DIF3);
    WRITELN('(7) STAGE 4 THRESHOLD:        ',THR4);
    WRITELN('(8) STAGE 4 DIFFERENCE:       ',REC_DIF4);
    WRITELN('(9) AVERAGING WEIGHT FACTOR: ',NUM);
    WRITELN;
  END;

{ ---------------------------------------------------------------------- }
```

```
PROCEDURE ADJUST;

{ ALLOWS AND OF THE ABOVE PARAMETERS TO BE ADJUSTED BY THE USER AT
  RUN TIME. }

  VAR SELECT : INTEGER;
        CH : CHAR;
    NEWTHR : INTEGER;

  BEGIN
    REPEAT
      WRITELN;
      WRITELN('(0) No change');
      DISPLAY_PARAMS;
      WRITE('Select parameter to adjust: ');
      READLN(INPUT,SELECT);
      CASE SELECT OF
        0 : ;          { Nothing done here }
        ELSE BEGIN
          WRITELN('NEW THRESHOLD: ');
          READLN(NEWTHR);
          CASE SELECT OF
            1 : FRM_THR:=NEWTHR;
            2 : THR1:=NEWTHR;
            3 : THR2:=NEWTHR;
            4 : REC_DIF2:=NEWTHR;
            5 : THR3:=NEWTHR;
            6 : REC_DIF3:=NEWTHR;
            7 : THR4:=NEWTHR;
            8 : REC_DIF4:=NEWTHR;
            9 : NUM:=NEWTHR;
          END;
        END;
      END;
    UNTIL SELECT = 0;          { Look for exit option }
    WRITELN;
  END;

{ ---------------------------------------------------------------------- }
PROCEDURE BANDSUM(VAR BDSUM:BAND_SAMPLE ;
                  VAR WDSUM:WS_SAMPLE);

{ PUTS INTO ARRAY BDSUM[I,J] THE TOTAL ENERGY OF BAND J, WORD I, FOR
  ALL SEVEN ENERGY BANDS FOR ALL THE LIBRARY WORDS.
  PUTS INTO ARRAY WDSUM[I] THE TOTAL ENERGY OF EACH WORD I, FOR ALL
  LIBRARY WORDS }

VAR   I,J,K   :  INTEGER;

BEGIN
  FOR I:=1 TO N_WORDS DO BEGIN
     FOR J:=1 TO N_BPF DO BEGIN
       BDSUM[I,J]:=0;
```

```
      FOR K:=1 TO WORDS[I].N DO
        BDSUM[I,J]:=BDSUM[I,J]+WORDS[I].DAT[K,J];
    END;
    WDSUM[I]:=0;
    FOR J:=1 TO N_BPF DO WDSUM[I]:=WDSUM[I]+BDSUM[I,J];
  END;
END;

{ ---------------------------------------------------------- }

PROCEDURE WAVEWRITER;

{ WRITES ALL THE WORD PATTERNS IN THE LIBRARY (ALL THE LIBRARY WORDS)
  BACK INTO THE LIBRARY FILE ON DISK. THE LIBRARY MAY BE CHANGED
  DURING THE PROGRAM SINCE WAVE AVERAGING AND ADDING WORDS TO THE LIBRARY
  MAY HAVE BEEN DONE.  ALL DATA STORED IN ASCII FORM. }

CONST   STCON = '0123456789ABCDEF';

VAR     N_FLE  : INTEGER;
        I,L,M  : INTEGER;
        ASKEY  : STRING[1];
        DUMB2,FF  : CHAR;
        HX16   : INTEGER;
        RM16   : INTEGER;

BEGIN
  FF:=CHAR(12);
  REWRITE(VOCAB);
  IF CMD15 = 'Y' THEN BEGIN
    WRITELN;
    WRITELN;
    WRITELN(LST,'UPDATED LIBRARY')
  END;
  WRITELN('WRITING UPDATED LIBRARY INTO FILE ',VOCAB_FILE);
  FOR I:=1 TO N_WORDS DO BEGIN
    WRITE(WORDS[I].TXT:8);
    IF CMD15 = 'Y'  THEN WRITELN(LST,WORDS[I].TXT);
    WRITELN(VOCAB,WORDS[I].TXT);
    FOR L:=1 TO WORDS[I].N DO BEGIN
      FOR M:=1 TO N_ELM DO BEGIN
        HX16:=TRUNC(WORDS[I].DAT[L,M]/16);
        RM16:=WORDS[I].DAT[L,M]-16*HX16;
        ASKEY:=COPY(STCON,HX16+1,1);
        WRITE(VOCAB,ASKEY:1);
        ASKEY:=COPY(STCON,RM16+1,1);
        WRITE(VOCAB,ASKEY:1);
      END;
    WRITE(VOCAB,NL);
    END;
    WRITE(VOCAB,'*',NL);
    WRITE(VOCAB,'$',NL);
  END;
```

```
  CLOSE(VOCAB);
  IF CHD15 = 'Y' THEN WRITE(LST,FF);

END;

{ ---------------------------------------------------------- }

PROCEDURE SET_FILES;

{ ALLOWS THE USER TO INPUT INTO ARRAY C_FILES TEN COMPARISON FILE
  NAMES WHICH ARE TO BE PROCESSED IN THE ORDER ENTERED. }

VAR     I : INTEGER;
        C_NAME : STRING[12];

BEGIN
  FOR I:=1 TO 10 DO C_FILES[I]:='';
  WRITELN;
  WRITELN('LIST THE DESIRED COMPARISON FILES,  * WHEN COMPLETE');
  C_NAME:='START';
  I:=1;
  READLN(C_NAME);
  WHILE (C_NAME <> '*') AND (I <> 11) DO BEGIN
    C_FILES[I]:=C_NAME;
    READLN(C_NAME);
    I:=I+1;
  END;
  WRITELN(I-1,'  FILES ENTERED');
END;

{ ---------------------------------------------------------- }

BEGIN  { Main program }
  INITIALIZE;
  BANDSUM(BDSUM,WDSUM);
  WRITELN;
  WRITELN('DEFAULT PARAMETERS ARE:');
  WRITELN;
  ADJUST;
  SET_FILES;
  FILENAME2:=C_FILES[1];
  C_INDX:=1;
  WHILE (FILENAME2 <> '') AND (C_INDX <> 11) DO BEGIN
    WRITELN;
    WRITELN('NEXT COMPARISON FILE IS ',FILENAME2,'.  NUMBER ',C_INDX);
    ASSIGN(VOCAB2,FILENAME2);
    RESET(VOCAB2);
    WIDX:=0;
    WHILE NOT EOF(VOCAB2) DO GO_RECOGN;
    C_INDX:=C_INDX+1;
    IF C_INDX <> 11 THEN FILENAME2:=C_FILES[C_INDX];
  END;
  IF RIGHTON = 1 THEN WAVEWRITER;
```

```
  WRITELN;
  WRITELN('MEMORY LEFT (16 BYTE PARAGRAPHS): ',MemAvail);

END.
```

REFERENCES

1. Liu, Thomas : A MICROCOMPUTER-BASED VOICE RECOGNITION SYSTEM, Master's Thesis, University of Illinois at Urbana-Champaign, 1983.

2. J. L. Flanagan, SPEECH ANALYSIS SYNTHESIS AND PERCEPTION. New York, N.Y.: Springer-Verlag, 1972.

3. A. V. Oppenheim, APPLICATIONS OF DIGITAL SIGNAL PROCESSING. Englewood Cliffs, N.J.: Prentice-Hall, 1978.

4. L. R. Rabiner and R. W. Schafer, DIGITAL PROCESSING OF SPEECH SIGNALS. Englewood Cliffs, N.J.: Prentice-Hall, 1978.

5. G. J. Vysotsky, "A speaker - independent discrete utterance recognition system, combining deterministic and probabilistic strategies," IEEE TRANS. ACOUST. SPEECH, SIGNAL PROCESSING, vol. ASSP-32, pp. 489-498, June 1984.

6. H. Helms et al., COMPUTER HANDBOOK. New York, N.Y.: McGraw Hill Book Co., 1983.

7. Zenith Data Systems, Z-100 PC SERIES COMPUTERS, 1984.

8. M. E. Van Valkenburg, ANALOG FILTER DESIGN. New York, N.Y.: CBS College Publishing, 1982.

9. National Semiconductor Corporation, LINEAR APPLICATIONS HANDBOOK, Vol. 2, 1977.

10. Intel Corporation, MICROCONTROLLER USER'S MANUAL, 1982.

11. Intel Corporation, COMPONENT DATA CATALOG, 1980.

12. National Semiconductor Corporation, LINEAR DATA BOOK, 1980.

13. Wigger, William : A DISCRETE UTTERANCE VOICE RECOGNITION ALGORITHM, Master's Thesis, University of Illinois at Urbana-Champaign, 1985.

14. R. J. Niederjohn and M. Lahat,"A zero-crossing consistency method for formant tracking of voiced speech in high noise levels," IEEE TRANS. ACOUST. SPEECH, SIGNAL PROCESSING, vol. ASSP-33, pp. 349-355, 1985.